IMPLEMENTATION OF PUBLIC KEY ELLIPTIC CURVE CRYPTOSYSTEMS

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

S. Kumar

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April 1996

CERTIFICATE

It is certified that the work contained in the thesis entitled " *Implementation of Public Key Elliptic Curve Cryptosystems*", by "*S.Kumar*", has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

30.4.96

(Dr. M.U.Siddiqi)
Professor
Dept. of Electrical Engg.
IIT Kanpur

April, 1996.

Abstract

Cryptography, as a means for sending secret information over insecure communication channels, is thousands of years old. Since the birth of public key cryptography in 1976, many public key cryptosystems have been proposed and many have been broken. Security of some of these schemes has been seriously threatened by the recent advances in computing discrete logarithms and integer factorisation.

Elliptic curve cryptosystems seem to be an efficient and viable alternative for the conventional systems. Security of these cryptosystems depends on the difficulty in finding discrete logarithm on an elliptic curve, called elliptic curve discrete logarithm problem (EDLP). Protocols implemented using elliptic curves have the advantage of having smaller keys than the existing systems for the same level of security.

In this thesis we have studied the issues involved in the implementation of cryptosystems, with specific reference to elliptic curve cryptosystems. Methods for optimising various computations in cryptosystems have been implemented and their timing details have been tabulated. The speed-up obtained by these optimisations has been demonstrated by implementing the well known RSA algorithm. A library for performing elliptic curve computations has been built. Finally elliptic curve analog of RSA and ElGamal schemes have been implemented and their throughput tabulated.

DEDICATION

To My Parents and Sisters

- *Who take pride in whatever little I do and make it a great achievement.*

To HIM

- *Who makes everything happen.*

ACKNOWLEDGEMENTS

I express my sincere gratitude to my thesis supervisor Dr. M. U. Siddiqi. First of all, I thank him for introducing me to the fascinating area of CRYPTOGRAPHY. It was his encouragement and foresight that made me take up this work after my initial resentment. His patience and tolerance, and the faith he had in me, helped me to work without pressure whenever I was nearing a deadline. Words are not enough to show my gratitude for his help and cooperation during the final stages.

I thank my parents and sisters who were of constant source of encouragement and who believed in me strongly and made me confident. My heartfelt gratitude is due to Dr. R. N. Biswas, the man who inspired me and was a turning point in my life. I thank The Head, ACES and DORD for permitting me to register for M. Tech and giving me this great opportunity. I thank Rajesh Chandy and M.R. Prabhu, my dearest friends from MSEC, for motivating me and guiding me in all my endeavours. My sincere thanks are due to Haripriya for helping me out of the toughest situations and lending me a helping hand. Without her support it would have been extremely difficult for me to cross many of the hurdles. I also thank Tanuja for the lively company, tasty food and all other helps. My thanks to Ganesan for his expert opinions and helps.

I thank Nagendra and Sandhya for helping me to bootstrap at IITK; especially to Sandhya and Navpreet for all their support. I thank Dr. K. R. Srivatsan for taking interest in me and for all the advice he had given me every now and then. My sincere thanks are also due to my teachers Dr. P. R. K. Rao, Dr. P. K. Chatterjee and Dr. V. Sinha. I thank Mr. Sethuraman, Dr. Udaya and Dr. Madhu of *CRL*[1], *Bangalore* for their valuable suggestions. Thanks to Dr. C. DasGupta for allowing me to use his Deskjet printer liberally.

I thank Dr. Don Coppersmith (*IBM T. J. Watson Res. Lab.*), Dr. Michael Pfiefer (*Universität des Saarlandes in Saarbrücken, Germany*), Dr. Alfred Menezes (*Univ. of Auburn*), Dr. Burt Kaliski (*RSA Labs.*), Dr. Ron Rivest (*RSA Labs.*), Dr. G. B. Agnew (*Univ. of Waterloo*) and Dr. Alan Sherman (*UMBC*) for their suggestions and comments. Thanx to the INTERNET which took me close to these great people. Special thanks to D. J. Delorie for making available his GNU ports for IBM PCs.

I take this opportunity to thank the technical officials, Mr. DVSSN Murty, Mr. S. S. Bhatnagar, Mr. A. C. Joshi, Mr. Kaushal, Mr. Tiwari, Mr. Ramnath and Mr.

[1]This work was partially funded by Central Research Lab., Bangalore through a project grant.

Balkishan for providing me various facilities whenever I had wanted them. I thank Mr. J. P. Gupta for his excellent, flawless typing work. Words of appreciation are not enough for Gomes, Ravi and DRDO Ravi, who helped me in typesetting this thesis. Special thanks to Prakash Veer for helping me in preparing transparencies and to Birlananth for the arrangements during the defence. I thank my uncle for providing me his PC during my stay in Madras. I thank Annath for helping me with laser print-outs. I thank all the Doordarshan Engineers, esply. Vadivazhagan and Patel, for their company and cooperation during the preparation of my thesis.

Special thanks to the following special people for their tuitions; Krishnamoorty, Burman, Mangal, Varada and Venu. Thanks to Kalyan Kuppuswamy, Kalyan Chakravarty and Pankaj Bansal for helping me with their expertise in Cryptography. Special thanks to Johnson for all his special help and an enjoyable company. My heartfelt thanx to all Hall4ites and Tamil friends who made my stay a memorable one, especially Kasi, Immie, Ilango, Raman, Malli, Naari, Pey Shankar, Gundu and olli Shankars, Srini, Arumugam, Kuppi, Palani, Sokku, Ramesh Rao, Ramesh Chandran and so many others.

Last but not the least, I owe a lot to the MONKEYs who gave me a great company in the lab during nights. They made me work hard by trapping me inside the lab and helping me finish the work in time.

If some names are missing here, it only means that it would be too small to mention their names here for their great helps.

Contents

List of Figures

Chapter 1

Introduction

Cryptography, as a means for sending secret information over insecure communication channels, is thousands of years old. Today communication networks are being extensively used by banks, industrial and government organisations to convey highly sensitive and privileged information. In this context, information security and authentication has become extremely important and much attention has been focussed onto this area.

A turning point in the field of cryptography came in 1976 when Diffie and Hellman came up with the notion of public key cryptography. It opened up a new era in cryptography with the possibility of secret communication without any transfer of secret keys. In the years that have followed, many public key cryptosystems have been proposed and many have been broken. RSA and ElGamal systems, which have stood the test of time, are worth mentioning. RSA is based on the difficulty of factoring large numbers and ElGamal is based on the difficulty of finding logarithms in finite fields. Security of these systems has been seriously threatened by the recent advances in computing discrete logarithms and integer factorisation [52].

Elliptic curves provide a viable and efficient alternative for the conventional systems. Set of rational points satisfying Weierstrass equation over finite fields, commonly known as elliptic curves over finite fields, provide a rich class of abelian groups. Elliptic curves over a finite field F_p or a ring Z_n can be applied to implement analogs of the Diffie-Hellman scheme, ElGamal scheme and RSA scheme, as well as primality testing and integer factorization. Cryptosystems based on elliptic curves, called elliptic curve cryptosystems, were first proposed by Koblitz and Miller ([31], [44]). The security of these systems depends on the difficulty in finding discrete logarithm on an elliptic curve called Elliptic Curve Discrete Logarithm Problem (EDLP). EDLP over a field K is harder than the DLP for the same K. Protocols implemented using elliptic curves have the advantage of having smaller keys than the existing systems for the same level of security. Thus elliptic curve cryptosystems are useful in applications like "smart cards" where both memory and processing power is limited. Smaller key length implies smaller data size and silicon area apart from

smaller bandwidth and memory requirements. Another advantage is that the users can choose different elliptic curves and still use the same hardware for performing underlying operations in the field over which the curves are defined.

By appropriately choosing the curve, one can have a secure cryptosystem on E/F_q, if the order of the curve is divisible only by a prime more than 30 digits [24]. It is conjectured that the low exponent attack on the RSA scheme cannot be analogously applied to the attack on the elliptic RSA scheme using a low multiplier ([36], [37]).

In cryptosytems like RSA, computations of the type $M^e (\text{mod } n)$ are common and have to be performed repeatedly. Modular arithmetic and exponentiations are extremely time consuming with the parameters used in the algorithm, which are integers, typically of size 200 digits. Hence an efficient technique to perform these computations speeds up the overall performance of the system substantially.

The basic operation performed on an elliptic curve is the computation of $d * P$, multiplicity of a point P on the elliptic curve modulo n, which corresponds to the computation of $M^e \text{ mod } n$. For a large n and e, the time complexity of elementary operations as well as the number of elementary operation are very high. Thus, reducing the number of such operations is important when implementing the above algorithms. Methods for efficient exponentiation have been applied to the computation of multiplicity of points.

In the present work we have studied the issues involved in the implementation of cryptosystems, with specific reference to elliptic curve cryptosystems. *Montgomery reduction* [45] has been implemented and *signed binary window method* [28] has been used for exponent reduction. Timing details have been tabulated. Implementation of cryptosystems over elliptic curves has been demonstrated through RSA and ElGamal schemes.

The thesis is organised as follows:

- Chapter 2 gives a brief overview of cryptography.
- Chapter 3 gives an overview of elliptic curves and elliptic curve cryptosystems.
- In chapter 4, issues involved in implementing a cryptosystem have been presented. MultiPrecision arithmetic, and modular computations have been discussed. Montgomery's method for modular reduction has been discussed in detail.
- Chapter 5 gives a detailed account of the various implementations done as part of this thesis. The performance of basic computations has been demonstrated by implementing the RSA algorithm. Elliptic curve cryptosystems have been demonstrated by implementing elliptic curve analog of RSA and ElGamal schemes and their throughput is tabulated.

Chapter 2

An Overview of Cryptography

2.1 Introduction

Cryptography ([15],[11], [34], [10], [18], [17]) is the art and science of securing information. It deals with the transformation of ordinary text, also known as plaintext, into an unintelligible form, known as ciphertext (CRYPTOGRAM), and vice versa. Transformation of plaintext into ciphertext is called encryption, and that of ciphertext into plaintext is called decryption. This is shown in Figure 2.1. Normally these transformations are controlled by one or more parameters referred to as keys. A key can take on one of many values. The range of possible values of the key is called the keyspace. A sender enciphers each message before transmission. The authorised receiver knows the appropriate deciphering function to apply to the received message to obtain the original message. An eavesdropper who intercepts the transmitted message gets only "garbage" (the ciphertext) which makes no sense to him since he does not know how to decrypt it.

Complementary to cryptography is cryptanalysis, the art and science of breaking ciphertext. Cryptology is the branch of mathematics embodying both cryptography and cryptanalysis.

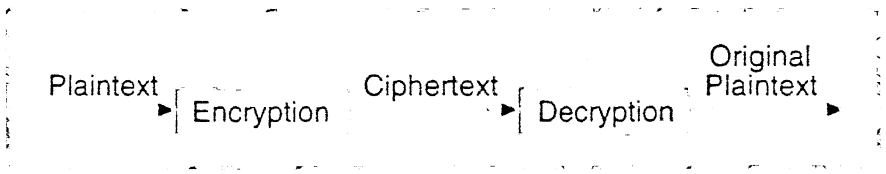

Figure 2.1: Encryption and Decryption

The motivation for encryption is secure transmission over insecure channels. Sender of a message should be assured that nobody can intercept and read or modify the message or be able to fabricate a realistic looking message.

Three of the most important services provided by cryptosystems are secrecy, authenticity, and integrity. Secrecy refers to denial of access to information by unauthorized individuals. Authenticity refers to validating the source of a message; i.e., that it was transmitted by a properly identified sender. Integrity refers to assurance that a message was not modified accidentally or deliberately in transit by replacement, insertion or deletion. Classical cryptography deals mainly with the secrecy aspect and it treats keys as secret. In later years two new trends emerged:

- The concept of public key.
- Authenticity.

The notion of public key arose from the difficulties traditionally associated with the management of secret keys. By making the keys public, the task of key management can be substantially simplified.

Authenticity has acquired great importance in the light of applications such as electronic mail systems and electronic funds transfers. In such settings, an electronic equivalent of handwritten signature is desirable. Also, intruders into a system often gain entry by masquerading as legitimate users; so an alternative to password systems is needed for access control. This leads to the concept of "Digital Signatures". An ideal system, which can solve all of these problems concurrently, providing both secrecy and authenticity, is Public Key Cryptosystem(PKC). Unfortunately no single technique proposed to date has met all the three criteria. Conventional systems such as DES require management of secret keys; systems using public key components may provide authenticity but are inefficient for bulk encryption of data due to low throughput.

Fortunately, conventional and public key systems are not mutually exclusive and they can complement each other. Public key systems can be used for signatures and also for the distribution of keys used in systems such as DES. Thus it is possible to construct hybrids of conventional and public key systems which can meet all of the above goals: secrecy,authenticity and ease of key management.

2.2 Requirements of a Cryptosystem

A cryptosystem has the following components:

- A plaintext message space, **M**.
- A ciphertext space, **C**.
- A key space, **K**.
- A set of enciphering transformations, $E_K : \mathbf{M} \rightarrow \mathbf{C}$, where $K \in \mathbf{K}$.

- A set of deciphering transformations, $D_K : \mathbf{C} \rightarrow \mathbf{M}$, where $K \in \mathbf{K}$.

Let E_K and D_K represent the encryption and decryption transformations respectively and E and D be the respective algorithms. It is always required that $D(E(M)) = M$ where M is the message. It may also be the case that $E(D(M)) = M$; in this event either E or D can be employed for encryption. It may be assumed that E_K and D_K are relatively easy to compute when K is known. The set of parameters describing E_K is called the enciphering key and that of D_K is the deciphering key.

Any cryptosystem must satisfy [15] the following requirements:

- The enciphering and deciphering transformations must be efficient for all keys.
- The system must be easy to use with a large keyspace. This discourages an exhaustive search.
- The security of the system should depend only on the secrecy of the keys and not on the secrecy of the algorithms.

2.2.1 Requirements for Secrecy

Secrecy requires that a cryptanalyst should not be able to determine the plaintext corresponding to given ciphertext, and should not be able to reconstruct D by examining ciphertext for known plaintext. This translates into two requirements [15] for a cryptosystem to provide secrecy:

- A cryptanalyst should not be able to determine M from $E(M)$; i.e. the cryptosystem should be immune to ciphertext-only attacks.
- A cryptanalyst should not be able to determine D given $\{E(M_i)\}$ for any sequence of plaintexts $\{M_1, M_2, \ldots\}$; i.e. the cryptosystem should be immune to known-plaintext attacks. This should remain true even when the cryptanalyst can choose $\{M_i\}$(chosen-plaintext attack), including the case in which the cryptanalyst can inspect $\{E(M_1), \ldots, E(M_j)\}$ before specifying M_{j+1}(adaptive chosen-plaintext attack).

It is to be noted that secrecy only ensures that decryption of a message by an intruder is infeasible. It does not imply authenticity or integrity.

2.2.2 Requirements for Authenticity and Integrity

Authenticity [15] requires that an intruder should not be able to masquerade as a legitimate user of a system. Integrity requires that an intruder should not be able to substitute false ciphertext for legitimate ciphertext. Following minimal requirements should be met for a cryptosystem to provide these services:

- It should be possible for the recipient of a message to ascertain its origin.
- It should be possible for the recipient of a message to verify that it has not been modified in transit.
- A sender should not be able to deny later that he sent a message.
- It should be possible for the recipient of a message to detect whether it is a replay of a previous transmission.

These requirements are independent of secrecy.

2.3 Classification

There are two fundamental classifications ([10], [18], [15]) of cryptosystems.

- Restricted use cryptosystems
- General use cryptosystems

A cryptosystem is restricted if its security relies on keeping the underlying algorithms secret. These systems provide inadequate security and only have historical significance. Breaking such systems is almost always a simple exercise for a professional cryptanalyst. These systems are of no relevance in the modern context. Security of a general cryptosystem depends on the secrecy of the key rather than the algorithm. It can be further classified into:

- Secret key or symmetric key cryptosystem.
- Public key or asymmetric key cryptosystem.

2.3.1 Secret Key Cryptosystem

In a secret key cryptosystem, E and D are parameterized by a single key K, so that we have $D_K(E_K(M)) = M$. In this system, both the sender and the receiver agree on a single, common key before actually communicating as shown in the Figure 2.2. This exchange of the key is through a secure medium. Since both the encryption

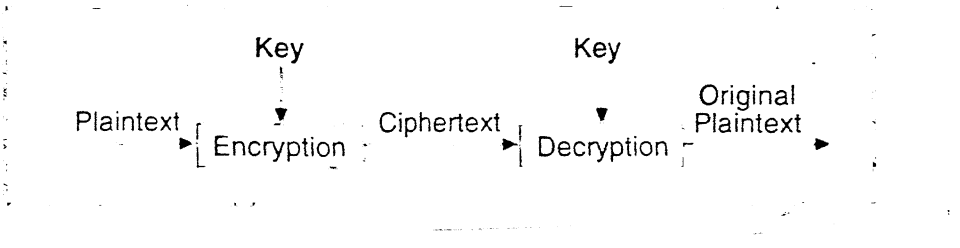

Figure 2.2: Symmetric Key Cryptosystem

and decryption keys are same, these systems are also known as symmetric key cryptosystems.

A typical secret key protocol will be as follows. Asha and Balu, who want to communicate secretly, decide on a secret key K. If Asha wants to send a message $m \in \mathbf{M}$ to Balu, she uses the enciphering algorithm E_K to produce $C = E_K(M)$ and sends C to Balu over an insecure channel. Balu uses the algorithm D_K to recover $m = D_K(C)$. Clearly, the security of the system depends on the secrecy of the key K. Secrecy and authenticity are both provided since an eavesdropper cannot compute $D_K(C)$ and a would-be masquerader cannot compute $E_K(M)$. In some cases (for e.g. transmission of a random bit string), this does not assure integrity; i.e. modification of a message enroute may be undetected. Typically integrity is provided by sending a compressed form of the message (a message digest) along with the full message as a check. In practice, the actual message is too long to be enciphered directly. So it has to be broken into pieces and enciphered repeatedly. If message is broken into groups of bits, called blocks, then that cipher system is called a *block cipher*. Some operate on the plaintext one bit at a time and are called *stream ciphers*.

The most notable example of a secret key cryptosystem is DES (Data Encryption Standard). It is a block cipher with a block size of 64 bits. We will not get into any more details of this system as we are more concerned about public key cryptosystems in this thesis.

2.3.2 Public Key Cryptosystem (PKC)

Secret key cryptosystems have an inherent problem known as the "key distribution problem" caused by the need for a common secret key. Before a private communication can begin, another private transaction is necessary to distribute corresponding encryption and decryption keys to the sender and receiver, respectively. Typically a secure medium is used to carry a key from the sender to the receiver. In this there is a danger that an unauthorized person might intercept and obtain these keys while

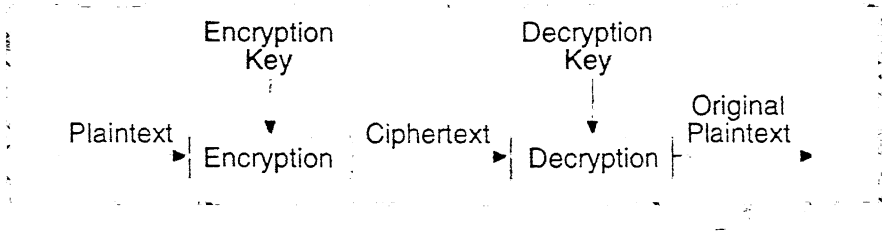

Figure 2.3: Asymmetric Key Cryptosystem

the keys are being communicated. In real complex sytems in which the keys are very large, maintaining the secrecy of the keys can be as troublesome as maintaining the secrecy of the text to be transmitted. Also, in case of a network, where a person needs to communicate to many persons, a large keyspace is required. Thus such a practice is not feasible if a communication system is to be rapid, secured and inexpensive.

In 1976, Diffie and Hellman [18] came up with the notion of public key cryptography. By having separate keys for encryption and decryption, public key (asymmetric key) cryptography provides both a mechanism for transmitting secret messages without prior exchange of a secret key and a method of implementing digital signatures.

Public key cryptography differs from conventional cryptography in the way the key is used. In case of PKC there are two different keys; the encryption key and the decryption key as shown in Figure 2.3. Hence the name two-key or asymmetric key cryptosystem. Each user places in a public file his encryption key (procedure) E. This public file is a directory giving the encryption key of each user. The user keeps secret the details of his corresponding decryption key (procedure) D. Thus each user can encrypt messages. However, without knowing the decryption key, messages sent by other users cannot be efficiently decrypted. Thus, this system provides a unique and powerful method of implementing a multi-user security system.
The procedures of PKC have the following four properties [49] :

(a) Deciphering the enciphered form of a message M yields M. i.e., $D(E(M)) = M$.

(b) Both E and D are easy to compute.

(c) By publicly revealing E the user does not reveal an easy way to compute D. This means that in practice only he can decrypt messages encrypted with E, or compute D efficiently.

(d) If a message M is first deciphered and then enciphered, M is the result. Formally, $E(D(M)) = M$.

An encryption (or decryption) procedure typically consists of a general method and an encryption key. The general method, under control of the key, enciphers a message M to obtain the enciphered form of the message, called the ciphertext C. Everyone can use the same general method; the security of a given procedure will rest on the security of the key. Revealing an encryption algorithm then means revealing the key.

When the user reveals E he reveals a very inefficient method of computing $D(C)$: testing all possible messages M until one such that $E(M) = C$ is found. If property (c) is satisfied the number of such messages to test will be so large that this approach is impractical.

A function E satisfying (a)-(c) is called a "trap-door one-way function"; if it also satisfies (d) it is a "trap-door one-way permutation". These functions are called "one-way" because they are easy to compute in one direction but very difficult to compute in the other direction. They are called "trap-door" functions since the inverse functions are in fact easy to compute once certain private "trap-door" information is known. Property (d) facilitates the implementation of "Digital Signatures". As an example let us examine how Balu sends a secret message M to Asha in a public key cryptosystem.

First, Balu retrieves E_A(public key of Asha) from the public file. Then he sends her the enciphered message $E_A(M)$. Asha deciphers the message by computing $D_A(E_A(M)) = M$. By property (c) of the public key cryptosystem, only she can decipher $E_A(M)$. Also she can encipher a private response with E_B, available in the public file. It is important to note that no private transactions between Asha and Balu are needed to establish private communication.

In 1978, Rivest, Shamir, and Adleman [49] of MIT published the first method of realizing public key cryptography. Their scheme, called the RSA system, is based on performing exponentiations in modular arithmetic. In the years that have followed, many public key cryptosystems have been proposed (and many have been broken). Currently, there are two kinds of systems that are considered viable and have been implemented:

- systems based on the difficulty of factoring the product of two large primes, and
- systems based on the difficulty in finding logarithms in a finite field (commonly known as discrete logarithm).

RSA system is the most popular of these public key systems and is potentially an extremely valuable cryptographic technique. Its security is based on the difficulty of factoring large numbers. ElGamal [23] proposed a public key cryptosystem and a signature scheme based on the discrete logarithm problem. We will briefly describe some of the encryption schemes in the following sections.

2.3.3 Diffie-Hellman Key Distribution Scheme

Diffie-Hellman [18] proposed the following scheme for key exchange. Suppose that Asha and Balu want to share a secret K_{AB}, where Asha has a secret x_A and Balu has a secret x_B. Let p be a large prime and α be a primitive element mod p, both public. Asha computes $y_A \equiv \alpha^{x_A} \bmod p$, and sends y_A. Similarly, Balu computes $y_B \equiv \alpha^{x_B} \bmod p$ and sends y_B. Then the secret K_{AB} is computed as

$$\begin{aligned} K_{AB} &\equiv \alpha^{x_A x_B} \bmod p \\ &\equiv y_A^{x_B} \bmod p \\ &\equiv y_B^{x_A} \bmod p. \end{aligned}$$

Hence both Asha and Balu are able to compute K_{AB}. But, for an intruder, computing K_{AB} is extremely difficult and is a discrete logarithm problem.

In any of the cryptographic systems based on discrete logarithms, p must be chosen such that $p-1$ has at least one large prime factor. If $p-1$ has only small prime factors, then computing discrete logarithms is comparatively easy.

2.3.4 ElGamal Scheme

In 1985, ElGamal[23] proposed a new encryption scheme, based on the implementation of the Diffie-Hellman key distribution scheme, that achieves a public key cryptosystem. The security of both the systems relies on the difficulty of computing discrete logarithms over finite fields.

In this scheme, each user i picks up a large prime p and a primitive element mod p say, α. He also chooses a random x_i between 0 and $p-1$ and computes

$$y_i \equiv \alpha^{x_i} \bmod p$$

.

y_i, α and p are made public and put in a public file. x_i is the secret key of the user i.

Now suppose that Asha wants to send Balu a message m, where $0 \leq m \leq p-1$. First Asha chooses a number k uniformly between 0 and $p-1$. Then she computes

$$K \equiv y_B^k \bmod p, \tag{2.1}$$

where y_B is the public key of Balu. The encrypted message (ciphertext) is then the pair (c_1, c_2), where

$$c_1 \equiv \alpha^k \bmod p \qquad \textit{and} \qquad c_2 \equiv Km \bmod p \tag{2.2}$$

Table 2.1: ELGamal Scheme

PUBLIC KEY:

p prime (can be shared among a group of users)

α $< p$ (can be shared among a group of users)

y_i $\equiv \alpha^{x_i} (\text{mod } p)$

PRIVATE KEY:

x_i $< p$

ENCRYPTION:

$0 \leq k \leq p-1$

$K \equiv y_B^k (\text{mod } p)$

$c_1 \equiv \alpha^k (\text{mod } p)$

$c_2 \equiv Km (\text{mod } p)$

DECRYPTION:

$K \equiv (\alpha^k)^{x_B} \equiv c_1^{x_B} (\text{mod } p)$

$m \equiv c_2/K \pmod{p}$

and K is as computed in 2.1.

The decryption operation splits into two parts. The first step is recovering K, which is easy for Balu since $K \equiv (\alpha^k)^{x_B} \equiv c_1^{x_B} \bmod p$, and x_B is known to Balu only. The second step is to divide c_2 by K and recover the message m. Table 2.1 summarizes the ElGamal scheme.

In this scheme the size of the ciphertext is double the size of the message. Also, the multiplication operation in 2.2 can be replaced by any other invertible operation such as addition mod$p$. The same value k should not be used for enciphering more than one block of the message, since if k is used more than once, knowledge of one block m_1 of the message enables an intruder to compute other blocks.

Due to the randomization in the enciphering operation, the cipher text for a given message m is not repeated. Also, due to the structure of this system, there is no obvious relation between the enciphering of m_1, m_2, and $m_1 m_2$, or any other simple function of m_1 and m_2. For the enciphering operation, two exponentiations are required. That is equivalent to about $2logp$ multiplications in $GF(p)$. For the deciphering operation only one exponentiation (plus one division) is needed.

Breaking the system is equivalent to breaking the Diffie-Hellman distribution scheme. First, if m can be computed from c_1, c_2, and y, then K can also be computed from y, c_1, and c_2(which appears like a random number since k and m are unknown). That is equivalent to breaking the distribution scheme. Second, (even if m is known) computing k or x from c_1, c_2, and y is equivalent to computing discrete logarithms. The reason is that both x and k appear in the exponent in y and c_1.

2.3.5 RSA Scheme

RSA Scheme [49] is an exponentiation cipher and involves modular exponentiation. The enciphering and deciphering transformations are based on Euler's generalisation of *Fermat's Theorem.* Let us look into some definitions([15], [47]) before discussing the RSA scheme.

Definition 1 *Given an integer n, $\phi(n)$ is the number of elements in the reduced set of residues modulo n, i.e., $\phi(n)$ is the number of positive integers less than n that are relatively prime to n.*

In general, for an arbitrary n, $\phi(n)$ is given by

$$\phi(n) = \prod_{i=1}^{t} p_i^{e_i - 1}(p_i - 1)$$

where

$$n = p_1^{e_1} p_2^{e_2} \cdots p_i^{e_i}$$

Fermat's Little Theorem 1 *Let n be a prime. Then for every a such that $gcd(a, n) = 1$,*

$$a^{n-1} \equiv 1 \pmod{n} \tag{2.4.2}$$

Euler provided a generalised version of the above theorem as follows.

Euler's generalisation 1 *For every a and n such that $gcd(a, n) = 1$,*

$$a^{\phi(n)} \equiv 1 \pmod{n} \tag{2.4.3}$$

RSA Procedure [49]
The encryption key is a pair of positive integers (e, n). Similarly, the decryption key is a pair of positive integers (d, n). Each user makes his encryption key public, and keeps the corresponding decryption key private.

Each user computes n as the product of two primes p and q, $n = p * q$. These primes are very large, "random" primes. Although n is public, the factors p and q are kept secret. This also hides the way d can be derived from e. d is chosen to be a large, random integer which is relatively prime to $\phi(n)$ where $\phi(n) = (p-1)*(q-1)$. That is, d satisfies:

$$\gcd(d, (p-1) * (q-1)) = 1$$

The integer e is computed from p, q, and d to be the "multiplicative inverse" of d, modulo $(p-1) * (q-1)$. Thus we have

$$e * d \equiv 1 (\text{mod } (p-1) * (q-1))$$

Table 2.2: RSA Encryption

PUBLIC KEY:
- n product of two primes, p and q (p and q must remain secret)
- e relatively prime to $(p-1) \times (q-1)$

PRIVATE KEY:
- d $e^{-1}(\text{mod } (p-1) \times (q-1))$

ENCRYPTING:
- $c = m^e(\text{mod } n)$

DECRYPTING:
- $m = c^d(\text{mod } n)$

Steps for encrypting a message M, using a public encryption key (e,n), where e and n are a pair of positive integers, is as follows. The message is represented as an integer between 0 and n-1. A long message is broken into a series of blocks and each block is represented as such as an integer. This converts the message blocks into numeric form necessary for encryption.

The message is encrypted by raising it to the e^{th} power modulo n. To decrypt the ciphertext, it is raised to another power d, modulo n. The encryption and decryption algorithms E and D are thus :

$$C \equiv E(M) \equiv M^e \pmod{n}, \quad \text{for a message M.}$$

$$M \equiv D(C) \equiv C^d \pmod{n}, \quad \text{for a ciphertext C.}$$

Table 2.2 summarizes the RSA scheme. It is to be noted that encryption does not increase the size of a message; both the message and the ciphertext are integers in the range 0 to n-1. Since $\phi(n)$ cannot be determined without knowing the prime factors p and q, it is possible to keep d secret even if e and n are made public. This means that the RSA scheme can be used for public key encryption, where the enciphering transformation is made public and the deciphering transformation is kept secret. The security of the system depends on the difficulty of factoring n into p and q.

2.3.6 Applicability and Limitations

The range of applicability of public key systems is limited in practice by its relatively low throughput, compared to their conventional counterparts. The low efficiency is

due to the computationally intensive encryption and decryption operations. This precludes the use of public key systems as replacements for conventional systems and makes their use for bulk data encryption infeasible, at least for the present. Inspite of these limitations, there are two major application areas for public key cryptosystems:

- Distribution of secret keys.
- Digital signatures.

No bulk encryption is needed when PKC is used to distribute keys, since the latter are generally short. Also, digital signatures are generally applied only to outputs of *hash functions*. These hash functions are public functions that maps a message of any length into a fixed-length value called *hash value*, which serves as an authenticator. In both the cases the data to be encrypted or decrypted is restricted in size. Thus the throughput (bits/sec) limitation of PKC is not a major limitation for either application.

Chapter 3

Elliptic Curves and Cryptosystems

3.1 Introduction

The set of points on an elliptic curve E defined over a field K form an abelian group. These provide a rich class of abelian groups making them attractive for cryptographic implementations. Koblitz and Miller[31] [44], proposed a variant of discrete log cryptography based on the elliptic curve discrete log problem (EDLP) in the group of points of an elliptic curve defined over a finite field. The security of these systems is based on the EDLP. These cryptosystems have two potential advantages over systems based on the multiplicative group of a finite field (and also over systems based on RSA).

- the great diversity of elliptic curves available to provide the groups; and
- the absence of subexponential time algorithms (such as those of 'index calculus' type) that could find discrete logs in these groups.

Recently, they have been used in devising efficient algorithms for factoring integers and for primality proving. In the field of cryptography, elliptic curves have found applications in the construction of public key cryptosystems and in the construction of pseudorandom bit generators and one-way permutations. Other uses of elliptic curves are found in coding theory, where they are used to obtain good error-correcting codes.

Elliptic curve cryptosystems potentially provide equivalent security as the existing public key schemes, but with shorter key lengths. Having short key lengths means smaller bandwidth and memory requirements and can be a crucial factor in some applications, for example the design of smart card systems.

3.2 Motivation

Although the discrete logarithm problem, as first employed by Diffie and Hellman in their public key exchange algorithm, referred explicitly to the problem of finding logarithms with respect to a primitive element in the multiplicative group of the field of integers modulo a prime p, this idea can be extended to arbitrary groups, with the difficulty of the problem varying with the representation of the group.

Consider a finite group G, and let a and b be elements of G. Then determining a value x such that $a^x = b$ is the discrete logarithm problem for G. The value for x is called as logarithm of b to the base a, and is denoted by $\log_a b$. The difficulty of determining this quantity depends on the representation of G. If the abstract cyclic group of order m is represented in the form of the integers modulo m, then the discrete logarithm problem reduces to the extended Euclidean algorithm. However, if $m+1$ is prime, and the group is represented in the form of the multiplicative group of the finite field F_{m+1}, the problem is much more difficult. If the group is represented as an elliptic curve group over a finite field, then the problem is again much more difficult. This makes elliptic curves a good candidate for cryptosystem design.

3.3 Review of Elliptic Curves

In this section we briefly review the Elliptic curves [38], [34], [51], [53].

Definition 2 *An elliptic curve $E(F_q)$ over a field F_q is defined as set of all points $(x,y) \in F_q \times F_q$ that are the solutions of the Weierstrass Equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{3.1}$$

where $a_i \in F_q$ for $i \in Z$ such that $E(F_q)$ is non-singular.

Let us define the following quantities

$$\begin{aligned} d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + 4a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\ c_4 &= d_2^2 - 24d_4 \end{aligned}$$

and

$$\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \tag{3.2}$$

$$j(E) = \frac{c_4^3}{\Delta} \tag{3.3}$$

The quantity Δ is called the *discriminant* of the weierstrass equation and $j(E)$ is called the *j-invariant* of E if $\Delta \neq 0$. The following theorems explain the significance of these quantities.

Theorem 1 *Let E be the given weierstrass equation(3.1). Then E is an elliptic curve, i.e., the weierstrass equation is non-singular if and only if $\Delta \neq 0$.*

Theorem 2 *Two elliptic curves $E_1(F_q)$ and $E_2(F_q)$ given by the non-singular weierstrass equations*

$$E_1 : y^2 + a_1'xy + a_3'y = x^3 + a_2'x^2 + a_4'x + a_6'$$

$$E_2 : y^2 + a_1''xy + a_3''y = x^3 + a_2''x^2 + a_4''x + a_6''$$

are isomorphic over F_q, denoted by $E_1 \cong E_2$, if and only if there exist variables $u, r, s, t \in F_q, u \neq 0$, such that the change of variables

$$(x, y) \longrightarrow (u^2x + r, u^3y + u^2sx + t)$$

transforms equation E_1 to equation E_2. The relation of isomorphism is an equivalence relation.

Properties of Elliptic Curves

Theorem 3 *If two elliptic curves $E_1(F_q)$ and $E_2(F_q)$ are isomorphic over F_q, then $j(E_1) = j(E_2)$. If two elliptic curves are isomorphic, then they are also isomorphic as abelian groups.*
The converse statement is not true in general.

Theorem 4 *(Hasse) Let $\#E(F_q)$ be the order of an elliptic curve group $E(F_q)$ defined over a field F_q with q elements. Then*

$$(\sqrt{q} - 1)^2 \leq \#E(F_q) \leq (\sqrt{q} + 1)^2$$

or

$$\#E(F_q) = q + 1 - t$$

where $q + 1$ is the expected number of solutions and t is the discrepency as (by Riemann Hypothesis for Abelian Variants of dimension 1)

$$|t| \leq 2\sqrt{q}$$

The curve $E(F_q)$ is said to be *supersingular* if $t^2 = 0, q, 2q, 3q, \text{or} 4q$ and *nonsupersingular* otherwise. Nonsupersingular curves are also referred to as ordinary curves. If the characteristic of F_q is 2 or 3, then a curve over F_q is supersingular if and only if it has j-invariant equal to zero. In case of characteristic > 3, the curve is supersingular iff its group has cardinality $q + 1$. The curve E can be viewed as an elliptic curve over any extention field F_{q^k} of F_q; $E(F_q)$.

Theorem 5 *(Cassels) The group $E(F_q)$ is either cyclic or the product of two cyclic groups of order m_1 and m_2 satisfying*

$$m_1|m_2, \qquad m_1|gcd(m, q-1)$$

where $m = \#E(F_q)$ and F_q has q elements.

Corollary *If $\#E(F_q)$ is squarefree, then surely $E(F_q)$ is cyclic.*
Remark *Similarly for the other case, if $\#E(F_q)$ is non-squarefree, then $E(F_q)$ need not be a cyclic group.*

For various characteristics of the field K denoted by $char(K)$, the weierstrass equation (3.1) can be simplified by means of coordinate transformation.
Elliptic Curves over K, char$(K) \neq 2, 3$ If $char(K) \neq 2$, then the following change of variables

$$(x, y) \longrightarrow (x, y - \frac{a_1 x}{2} - \frac{a_3}{2})$$

transform E given by equation(3.1) to E' over K, where $E \cong E'$ as below.

$$E' : y^2 = x^3 + b_2 x^2 + b_4 x + b_6$$

where

$$\begin{aligned} b_2 &= a_2 + \frac{a_1^2}{4} \\ b_4 &= a_4 + \frac{a_1 a_3}{2} \\ \text{and} b_6 &= a_6 + \frac{a_3^2}{4}. \end{aligned}$$

If $char(K) \neq 2, 3$, then by further change of variables

$$(x, y) \longrightarrow (\frac{x - 3b_2}{36}, \frac{y}{216})$$

we get,

$$E'' : y^2 = x^3 + ax + b \tag{3.4}$$

$E' \cong E''$ and hence $E \cong E''$. The above equation is nothing but a weierstrass equation where $a_1 = a_2 = a_3 = 0$, and is referred to as *weierstrass short normal form.* Applying this to the equations (3.2, 3.3) we get the following quantities specialised for the equation (3.4).

$$\Delta = -16(4a^3 + 27b^2), \quad \text{and}$$

$$j(E) = \frac{-1728(4a)^3}{\Delta}$$

Since E is non-singular, $\Delta \neq 0$ and so we have the condition

$$4a^3 + 27b^2 \neq 0$$

from above.

Elliptic Curves over K, char(K) = 2

Consider E, defined over a field K with characteristic 2, given by the general Weierstrass equation,

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Using equation (3.3), we calculate the j-invariant for $char(K) = 2$ as

$$j(E_k) = \frac{a_1^{12}}{\Delta}$$

Depending on $j(E)$ there are two kinds of curves defined.

- **j-invariant equals zero**
  Since $j(E) = 0$, we have $a_1 = 0$. By the following transformation

  $$\theta : (x, y) \longrightarrow (x + a_2, y)$$

  we get

  $$E_1 : y^2 + a_3'y = x^3 + a_4'x + a_6'$$

  where $E(K) \cong E_1(K)$. From equations (3.2, 3.3) we obtain $\Delta = {a_3}^4$ and $j(E_1) = 0$.

- **j-invariant not equal to zero**
  Consider the following transformation

  $$\xi : (x, y) \longrightarrow (a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2 + a_3^2}{a_1^3})$$

  We get

  $$E_2 : y^2 + xy = x^3 + a_2x^2 + a_6$$

  From equations (3.2, 3.3) we obtain $\Delta = a_6$ and $j(E_2) = \frac{1}{a_6}$.

3.3.1 Elliptic Curve Group Structure

The points on an elliptic curve alongwith a special point $\mathcal{O}$ called as *point at infinity* (the identity element) form an abelian group under certain addition operation. Let E be an elliptic curve defined by the weierstrass equation (3.1). Consider two points $P, Q \in E$. Then, addition is defined as follows:

- a) $\mathcal{O} + P = P$ and $P + \mathcal{O} = P$.($\mathcal{O}$ is the identity element)
- b) $-\mathcal{O} = \mathcal{O}$.
- c) If $P = (x_1, y_1) \neq \mathcal{O}$, then $-P = (x_1, -y_1 - a_1x_1 - a_3)$.

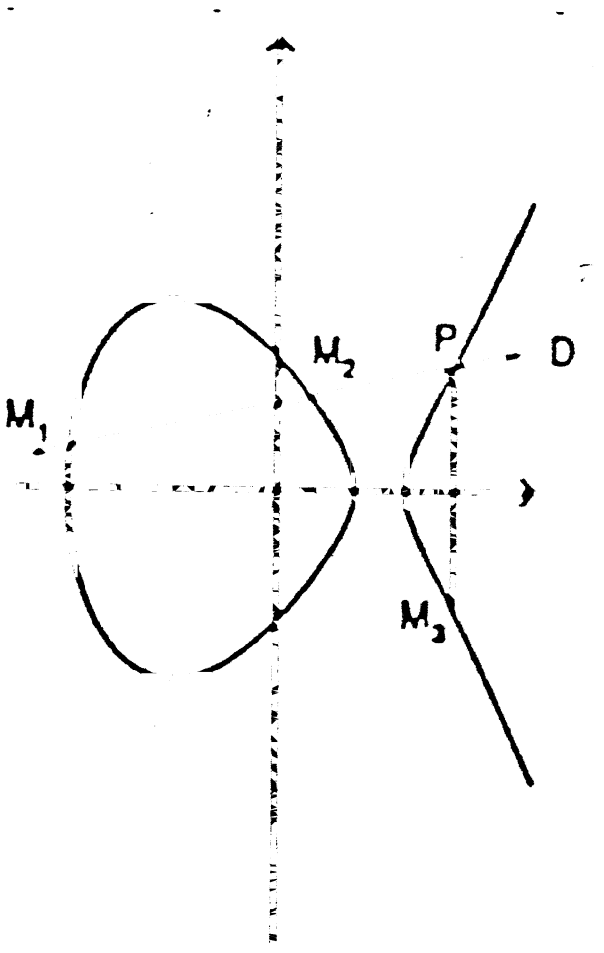

Figure 3.1: An elliptic curve over $\mathcal{R}$.

- d) If $Q = -P$, then $P + Q = \mathcal{O}$.
- e) If $P \neq \mathcal{O}, Q \neq \mathcal{O}, Q \neq -P$, then let R be the third point of intersection of either the line PQ if $P \neq Q$ or the tangent line to the curve at P if $P = Q$, with the curve. Then, $P + Q = -R$.

As in any abelian group, the notation nP denotes P added to itself n times if n is positive, and $-P$ added to itself $|n|$ times if n is negative, and $0P = \mathcal{O}$. Explicit formulae for the group operation $+$ is defined as following. Consider $P = (x_1, y_1), Q = (x_2, y_2)$ and let $P + Q = (x_3, y_3)$ be points on the general elliptic curve equation given below

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

Define

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\ \dfrac{3x_1^2 + 2a_2x_1 + a_4 - a_1y_1}{2y_1 + a_1x_1 + a_3}, & \text{if } P = Q. \end{cases}$$

Let $\varphi = y_1 - \lambda x_1$. Then,

$$x_3 \stackrel{\text{def}}{=} \lambda^2 + a_1\lambda - a_2 - x_1 - x_2$$

$$y_3 \stackrel{\text{def}}{=} -(\lambda + a_1)x_3 - \varphi - a_3$$

Figure (3.1) shows the geometric interpretation of addition of points. For two points M_1 and M_2 on the curve where $M_1 \neq M_2$, the sum $M_3 = M_1 + M_2$ is the mirror image of the third point P on the curve where the line joining M_1 and M_2 again meets the curve. If $M_1 = M_2$ then the tangent line is used. The line joining the points M_1 and M_2 has a gradiant λ given by the above formula; the alternative is derived from the limiting case when the chord becomes the tangent at M_1. This line intersects the curve at one further point P, whose negative is defined to be the sum of M_1 and M_2. One way of interpreting the addition law on $E(F_q)$ is to state that the three points P, Q, R are colinear if and only if

$$P + Q + R = \mathcal{O}$$

Addition Formulae

Elliptic curve over field K, char$(K) > 3$
Consider the elliptic curve

$$E : y^2 = x^3 + ax + b$$

If $P = (x_1, y_1) \in E$, then $-P = (x_1, -y_1)$ and $P + (-P) = \mathcal{O}$. Given another point $Q = (x_2, y_2) \in E$, it satisfies the above properties and if $Q \neq -P$, then $P + Q = (x_3, y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2 \tag{3.5}$$

$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{3.6}$$

$$\tag{3.7}$$

where

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if } P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1}, & \text{if } P = Q \end{cases}$$

Elliptic curve over field K, char$(K) = 2$

- zero j-invariant elliptic curves ($j(E) = 0$).

  For this case the elliptic curve equation can be written as

$$E : y^2 + a_3 y = x^3 + a_4 x + a_6$$

  and the addition formula is given as follows.

  If $P = (x_1, y_1) \in E$, then $-P = (x_1, y_1 + a_3)$ and $P + (-P) = \mathcal{O}$. Given another point $Q = (x_2, y_2) \in E$, it satisfies the above properties and if $Q \neq -P$, then $P + Q = (x_3, y_3)$ where

$$x_3 = \begin{cases} \left(\dfrac{y_2 + y_1}{x_2 + x_1}\right)^2 + x_1 + x_2, & \text{if } P \neq Q \\ x_1^4 + a_4^2, & \\ a_3^2, & \text{if } P = Q \end{cases}$$

  and

$$y_3 = \begin{cases} \left(\dfrac{y_2 + y_1}{x_2 + x_1}\right)(x_1 + x_3) + y_1 + a_3, & \text{if } P \neq Q \\ \left(x_1^2 + a_4\right)(x_1 + x_3) + y_1 + a_3, & \text{if } P = Q \\ a_3 & \end{cases}$$

- non-zero j-invariant elliptic curves ($j(E) \neq 0$).

  For this case, the elliptic curve equation can be written as

$$E : y^2 + xy = x^3 + a_2 x^2 + a_6$$

  and the addition formula is given as follows.

  If $P = (x_1, y_1) \in E$, then $-P = (x_1, y_1 + a_3)$ and $P + (-P) = \mathcal{O}$. Given another point $Q = (x_2, y_2) \in E$, it satisfies the above properties and if $Q \neq -P$, then $P + Q = (x_3, y_3)$ where

$$x_3 = \begin{cases} \left(\dfrac{y_2 + y_1}{x_2 + x_1}\right)^2 + \dfrac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a_2, & \text{if } P \neq Q \\ x_1^2 + \dfrac{a_6}{x_1^2}, & \text{if } P = Q. \end{cases}$$

  and

$$y_3 = \begin{cases} \left(\dfrac{y_2 + y_1}{x_2 + x_1}\right)(x_1 + x_3) + y_1 + x_3, & \text{if } P \neq Q \\ x_1^2 + \left(x_1 + \dfrac{y_1}{x_1}\right)x_3 + x_3, & \text{if } P = Q \end{cases}$$

3.4 Elliptic Curves over a Ring Z_n

We now consider elliptic curves over the ring Z_n ([38],[35]), where n is an odd composite squarefree integer. An elliptic curve $E_n(a,b)$ can be defined as the set of pairs $(x,y) \in Z_n^2$ satisfying $y^2 \equiv x^3 + ax + b(\text{mod } n)$, together with a point $\mathcal{O}$ at infinity. An addition operation on $E_n(a,b)$ can be defined in the same way as the addition operation on $E_q(a,b)$, simply by replacing computations in F_q by computations in Z_n. However, two problems occur. The first problem is that because the computation of λ requires a division which in a ring is defined only when the divisor is a unit, the addition operation on $E_n(a,b)$ is not always defined. The second problem, which is related to the first is that $E_n(a,b)$ is not a group. It would therefore seem impossible to base a cryptographic system on $E_n(a,b)$. However, it is still possible to do so for the following reasons.

Let $n = pq$ be the product of only two primes as in the RSA system. Moreover, the addition operation on $E_n(a,b)$ described above, whenever it is defined, is equivalent to the (componentwise defined) group operation on $E_p(a,b)xE_q(a,b)$. By the Chinese Remainder Theorem, every element c of Z_n can be represented uniquely as a pair $[c_p, c_q]$ where $c_p \in Z_q$. Thus every point $P = (x,y)$ on $E_n(a,b)$ can be represented uniquely as a pair $[P_p, P_q] = [(x_p, x_p), (x_q, y_q)]$ where $P_p \in E_p(a,b)$ and $P_q \in E_q(a,b)$, with the convention that $\mathcal{O}$ is represented by $[\mathcal{O}_p, \mathcal{O}_q]$, where $\mathcal{O}_p$ and $\mathcal{O}_q$ are the points at infinity on $E_p(a,b)$ and $E_q(a,b)$, respectively . By this mapping, all elements of $E_p(a,b)xE_q(a,b)$ are exhausted except the pairs of points $[P_p, P_q]$ for which exactly one of the points P_p and P_q is the point at infinity. Note that the addition operation on $E_n(a,b)$ described above is undefined if and only if the resulting point, when interpreted as an element of $E_p(a,b) \times E_q(a,b)$, is one of these special points.

It is important to note that when all prime factors of n are large, it is extremely unlikely that the sum of two points on $E_n(a,b)$ is undefined. In fact, if the probability of the addition operation being undefined were non-negligible, then the very execution of a computation on $E_n(a,b)$ would be a feasible factoring algorithm, which is assumed not to exist. Therefore, the first problem will cause no difficulties in practice.

The second problem, that $E_n(a,b)$ is not a group, can be solved by the following lemma. That is, although we cannot use the properties of a finite group directly, we can use a property of $E_n(a,b)$ which is similar to that of a finite group. The following lemma [35] can be easily obtained from the Chinese Remainder Theorem.

Lemma Let $E_n(a,b)$ be an elliptic curve such that $\gcd(4a^3 + 27b^2, n) = 1$ and $n = pq(p,q\text{: prime})$. Let N_n be $\text{lcm}(\#E_p(a,b), \#E_q(a,b))$. Then, for any $P \in E_n(a,b)$, and any integer k,

$$(\text{k.N}_n + 1).P = P \text{ over } E_n(a,b).$$

We should note that it is possible to define an elliptic curve over a ring so that the

resulting structure is a group.

3.5 Implementations over F_{2^m}

From the addition formulae, it can be seen that two distinct points on an elliptic curve can be added by means of three multiplications and one inversion of field elements in the underlying field K, while a point can be doubled in one inversion and four multiplications in K. Additions and subtractions are not considered in this count, since these operations are relatively inexpensive. We have to select a curve and field K so as to minimize the number of field operations involved in adding two points.

Curves over $K = F_{2^m}$ have some specific properties which make them attractive for the implementation of cryptosystems. The field F_{2^m} can be viewed as a vector space of dimension m over F_2. That is, there exists a set of m elements $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in F_{2^m} such that each $\alpha \in F_{2^m}$ can be written uniquely in the form

$$\alpha = \sum_{i=0}^{m-1} a_i \alpha_i, \quad \text{where } a_i \in \{0, 1\}.$$

We can then represent α as the 0-1 vector $\{a_0, a_1, \ldots, a_{m-1}\}$. In hardware, field element is stored in a shift register of length m. Addition of field elments is performed by bitwise XOR-ing the vector representations, and takes one clock cycle. If the field elements are represented in a special basis, called *normal basis*, then squaring is just a rotation operation. But multiplication in a normal basis is quite complex. However, it is possible to choose a normal basis, called *optimal normal basis*, and the complexity can be reduced substantially. Curves over F_{2^m} with zero j-invariant are advantageous for the following reasons.

- The arithmetic in F_{2^m} is easier to implement in computer hardware than the arithmetic in finite fields of characteristic greater than 2.
- When using a normal basis representation for the elements of F_{2^m}, squaring a field element becomes a simple cyclic shift of the vector representation, and thus reduces the multiplication count in adding two points.
- For curves of zero j-invariant over F_{2^m}, the inverse operation in doubling a point can be eliminated by choosing $a_3 = 1$.
- For the curves for which m is odd, it is easy to recover the y-coordinate of a point given its x-coordinate and a single bit of the y-coordinate. This is useful in message imbedding, and in reducing message expansion in the ElGamal scheme.

If a normal basis representation is chosen for the elements of F_{2^m}, we see that doubling a point in E is "free", while adding two distinct points in E can be accomplished in two multiplications and one inverter. For the reasons given above, curves over F_{2^m} are preferred for hardware implementations like "Smart Card".

3.5.1 Projective Coordinates

From the addition formulae, we see that adding two distinct points on a *non-supersingular curve* over F_{2^m} takes 2 field multiplications and 1 inversion, while doubling a point takes 3 and 1 respectively. Even though there are special techniques for computing inverses in F_{2^m}, an inversion is still far more expensive than a field multiplication. The inverse operation needed when adding two points can be eliminated by resorting to projective coordinates.

Let E be the nonsupersingular curve $y^2 + xy = x^3 + a_2x^2 + a_6$. over $K = F_{2^m}$. E can be equivalently viewed as the set of all points in $P^2(K)$ which satisfy the homogeneous cubic equation $y^2z + \mathrm{xyz} = x^3 + a_2x^2z + a_6z^3$. Here $P^2(K)$ denotes the projective plane over K. The points of $P^2(K)$ are all of the non-zero triples in K^3 under the equivalence relation $\sim$, where $(x, y, z) \sim (x', y', z')$ if and only if there exists $\alpha \in K^*$ such that $x' = \alpha x, y' = \alpha y$ and $z' = \alpha z$. The representative of an equivalence class containing (x, y, z) will be denoted by $(x : y : z)$. It is to be noted that the only projective point in E with z-coordinate equal to 0 is the point $(0 : 1 : 0)$ which is the point at infinity $\mathcal{O}$ of E. If $\mathcal{O} \neq (x : y : z) \in E$, then $(x : y : z) = (x/z : y/z : 1)$, and so the projective point $(x : y : z)$ corresponds uniquely to the affine point $(x/z, y/z)$.

Let $P = (x_1 : y_1 : z_1) \in E$, $Q = (x_2 : y_2 : 1) \in E$, and suppose that $P, Q \neq \mathcal{O}, P \neq Q$ and $P \neq -Q$. Since $P = (x_1/z_1 : y_1/z_1 : 1)$ we can use the addition formula for E in affine coordinates to find $P + Q = (x_3\prime : y_3\prime : 1)$. We obtain

$$\begin{aligned} x_3\prime &= \frac{B^2}{A^2} + \frac{B}{A} + \frac{A}{z_1} + a_2, \\ y_3\prime &= \frac{B}{A}\left(\frac{x_1}{z_1} + x_3\prime\right) + x_3\prime + \frac{y_1}{z_1}, \end{aligned}$$

where $A = (z_1x_2 + x_1)$ and $B = (z_1y_2 + y_1)$.

To eliminate the denominators of the expressions for $x_3\prime$ and $y_3\prime$, we set $z_3 = A^3z_1$, $x_3 = x_3\prime z_3$ and $y_3 = y_3\prime z_3$, to get $P + Q = (x_3 : y_3 : z_3)$, where

$$\begin{aligned} x_3 &= AD \\ y_3 &= CD + A^2(Bx_1 + Ay_1) \\ z_3 &= A^3z_1, \end{aligned}$$

and where $C = A + B$, and $D = A^2(A + a_2 z_1) + z_1 BC$. From the above formulae we can conclude the following. Using projective coordinates, for addition, we need 13 multiplications as compared to 2 in affine coordinates. However, we avoid the costly inversion. Similarly a doubling requires 7 multiplications which is better than 3 multiplications and 1 inversion.

3.6 Security Aspects of Elliptic Curves

3.6.1 Elliptic Curve Discrete Logarithm Problem

Elliptic curves can be split into two classes, namely those which are supersingular, and those that are not. In [43] it is shown that for purposes of solving the discrete logarithm problem, these groups could be mapped isomorphically into the multiplicative group of an extension of the underlying field. Under this mapping, the security of systems based on the two classes of curves differs radically. There are two types of algorithms for solving (attacking) the discrete logarithm problem[43], [38], namely, general attacks which are independent of the representation of the underlying group, and specific attacks, which depend on the representation as mentioned earlier.

In terms of elliptic curves, the discrete logarithm problem, referred to as the elliptic discrete logarithm problem (**EDLP**), is the following. Let $E(F_q)$ be an elliptic curve over F_q and let P be a point in $E(F_q)$. For any point $R \in < P >$ (the subgroup generated by P), determine an efficient method to find the integer $k, 0 \leq k \leq \#P - 1, (\#P$ is the order of $P)$ such that $kP = R$.

The most powerful *general* algorithm, for solving discrete logarithm problem, known at present is the baby-step giant-step algorithm of Shanks. Let G be a group of order n and consider the interval $I(n)$ of integers from 0 to $n-1$. Let α and β be two members of G, and suppose we want to determine (if such exists), an integer x in $I(n)$ such that $\alpha^x = \beta$. The algorithm is as follows. Let $m = \lceil n \rceil$. Then precompute a list of pairs (i, α^i) for $0 \leq i \leq$ m. Then for each $j, 0 \leq j \leq m$, compute $\beta\alpha^{-jm}$, and see if this element is the second component of a member of the precomputed list. If $\beta\alpha^{-jm} = \alpha^i$ for some $i, 0 \leq i \leq m$, then $\beta = \alpha^{i+jm}$, otherwise no solution exists for x. Algorithms in this class have running times no better than $O(\sqrt{P})$, where p is the largest prime dividing the order of G.

The most successful attack on the elliptic logarithm problem so far is a method due to Menezes, Okamoto, and Vanstone[43], known as MOV attack. They showed that the discrete log problem on an elliptic curve can be reduced to (i.e. has the same complexity as) the discrete log problem in a finite field. This results in a subexponential algorithm for EDLP in case of supersingular curves making them a bad choice for cryptosystem designs.

Let E be an elliptic curve over $\bar{F}_q$, the algebraic closure of $F_q \cdot E(F_q)$ is the set of all points in E with coordinates from $F_q \cdot E(F_q)$ has finitely many points, whereas

E has infinitely many. Define

$$E[n] = \{P \in E : nP = 0\}.$$

$E[n]$ is called the set of n-torsion points of E. Now for each n, $\gcd(n, q) = 1$, there exists a positive integer k such that $E[n] \subseteq E(F_q k)$ and an isomorphism from $E[n]$ to a subgroup of $F_q k$ can be computed using the Weil pairing [51]. There exists a random polynomial time algorithm for computing the Weil pairing. These results form the basis for the MOV attack.

Let $E(F_q)$ be an elliptic curve over F_q and let P be a point of order n(i.e. $\# < P >= n$). To apply the MOV method, if $\gcd(n, q) = 1$, determine the smallest value of k such that $E[n] \subseteq E(F_q k)$. It is to be noted that a necessary condition for $E[n] \subseteq E(F_q k)$ is that $n|q^k - 1$. Now R is a point of $E(F_q)$ whose log with respect to P is s. This logarithm can be found by using the index calculus method for F_q^*k. Thus even though the index calculus methods do not apply directly to $E(F_q)$, we map a subgroup of this group into an algebraic structure where the method does apply.

MOV show that, in the case of supersingular curves, MOV becomes a subexponential attack. This happens because it can be shown that all supersingular curves have very small values of k associated with them. In general, however, nonsupersinguar curves have large values of k associated with them. If $k > \log^2 q$, then the index calculus method is $F_q k$ become fully exponential and the MOV attack is worse than the square root attacks. For an ordinary elliptic curve with q being a large prime, there exists an n-torsion subgroup on which the weil pairing can't be defined. This implies that EDLP on E cannot be reduced to DLP in any extension field by any embedding for these curves. Miyaji [7] studied EDLP on these curves and showed that EDLP E can be reduced to EDLP, to which the MOV reduction is possible. Summary of this study is as below.

- For any elliptic curve E defined over F_{2^m}, we can reduce EDLP on E to EDLP, to which the MOV reduction is applicable in an expected polynomial time.

- For a certain ordinary elliptic curve E defined over F_p, there exists EDLP on E which which makes any embedding to DLP in any extension field of F_p inapplicable. Then such EDLP on E/F_p is secure enough for all known attacks.

Consider a nonsupersingular curve $E(F_q)$ where $q = 2^{155}$. It is known [51] that $E(F_q) \cong Z_{n1} \times Z_{n2}$ where $n_2|n_1$. Suppose also that p is a prime dividing n_1 and that p has about 40 decimal digits. For an elliptic curve cryptosystem to be secure against the first attack, the order of the curve $\#E(F_q)$ has to contain a large prime factor. It is possible to find curves over F_q whose order is divisible by a prime factor with upto 46 decimal digits. These curves are secure with present computational skills and algorithms available. If the smallest value of k for which $E[n_1] \subseteq E(F_q k)$ is at

least 10, then the MOV attack requires an index calculus attack in a field with more than 1500 bits. For this elliptic curve, the most efficient way to compute elliptic logarithms is by one of the square root attacks which is infeasible for numbers of this size.

3.6.2 Cryptographic Implications

La Macchia and Odlyzko have recently implemented the Gaussian integer variant of the index calculus method, and they were easily able to compute logarithms in F_p, p a 192-bit prime. While the number field sieve has a much better asymptotic running time than the Gaussian integer method, it does not seem to be practical for fields F_p, where $p \leq 2^{512}$. For F_{2^m}, recent computations of Gordon and McCurley indicate that computing logarithms in F_{2^m} for m about 500 is barely feasible given large amounts of computer resources. Therefore it appears that, given the best algorithms known for the discrete logarithm problem in finite fields and given the best available computer technology, the discrete logarithm is intractable for finite fields of size greater than 2^{600}.

3.6.3 A Practical Implementation

It is worth mentioning that the use of nonsupersingular curves in public key cryptography provides by far the greatest security per bit of any known public key system. It was believed that elliptic curves could not be implemented efficiently, and therefore would be of little practical use. But the research group at University of Waterloo, has gone the farthest in improving and implementing elliptic curve cryptography in VLSI. They have developed an arithmetic processor [3] in a field $F_{2^{155}}$ which allows an efficient and practical implementation of elliptic curve cryptosystems.

In [3], for supersingular curves over $F_{2^{310}}$, the estimated throughput rate is reported as approximately $44 * 10^3$ bits per second. For the nonsupersingular case, assuming a hamming weight of 20 and a clock rate of 40 MHz, the approximate throughput rate on any nonsupersingular curve over $F_{2^{155}}$ is given as $\approx 60 * 10^3$ bits per second. For an unrestricted hamming weight, the approximate throughput is 40 kbits per second.

3.7 Elliptic Curve Cryptosystems

In the following subsections, we shall discuss the elliptic curve analogs of some public key cryptosystems discussed earlier.

3.7.1 Diffie-Hellman Scheme

We describe bellow the elliptic curve analog of Diffie-Hellman Scheme [34]. Suppose that Asha and Balu want to agree upon a key which will later be used in conjunction with a classical secret key cryptosystem. They first publicly choose a finite field F_q and an elliptic curve E defined over it. Their task is to choose a point P_K in such a way that all of their communication with one another is public and yet no one other than the two of them knows what P_K is.

Asha and Balu publicly choose a point $P_B \in E$ to serve as their *"base point"*. P_B plays a role similar to the generator in the finite-field Diffie-Hellman system. However, in this case P_B need not be a generator of the group of points on E. It is enough if P_B is a fixed publicly known point on E whose order is very large such that it is either N or a large divisor of N, where N is the order of the curve.

To generate a key, Asha chooses a random integer a of order of magnitude q which she keeps secret. She computes $aP_B \in E$, which she makes public. Similarly Balu chooses a random b and makes public $bP_B \in$ E. The secret key is then $P_K = abP_B \in E$. Both the users can now easily compute the key P_K. For example, in case of Asha, a is her secret key and she knows bP_B since it is public. However, a third party knows only aP_B and bP_B. Without solving the discrete logarithm problem there is no way to compute abP_B knowing only aP_B and bP_B.

3.7.2 ElGamal Scheme

Koblitz suggested a procedure for implementing ElGamal Scheme [34] over elliptic curves. Consider an elliptic curve $E(F_q)$ defined over a field F_q with order $\#E(F_q)$. Let $P_B \in E(F_q)$ be a fixed and publicly known point called the Base Point. Each user chooses an integer ϑ_i randomly, such that, $0 < \vartheta_i < \#E(F_q)$ and makes the point $\vartheta_i P_B$ public while keeping ϑ_i secret.

Message Imbedding

Before encrypting, messages have to be related to points on the working curve. This is called the *Message Imbedding*. Lengthy messages are made into blocks and each block is suitably associated to a point on the curve. This has to be done in a simple systematic way, so that the plaintext m, which is an integer in some range can readily be determined from the knowledge of the coordinates of the corresponding point P_m. The plaintext imbedding is not the same thing as encryption, but enables an authorized user of the system to recover m after deciphering the ciphertext point.

It is to be noted that there is no polynomial time deterministic algorithm known for writing down a large number of points on an arbitrary elliptic curve E over F_q. However, there are probabilistic algorithms for which the chance of failure is very small. Also, it is not enough to generate random points of $E(F_q)$; we need a systematic way to generate points that are related to m in some way. We describe

below a probabilistic method [34], which we have implemented, to imbed plaintexts as points on an elliptic curve E defined over F_q, where $q = p^r$ is assumed to be large.

Let k be a large enough integer so that we are satisfied with a failure probability of 1 out of 2^k. Let us assume that our message units m are integers $0 \leq m < M$ and our finite field is chosen so that $q > Mk$. We write the integers from 1 to Mk in the form $mk + j$, where $1 \leq j \leq k$. Thus we set up a one-to-one correspondence between such integers and a set of elements of F_q. Given m, for each $j = 1, 2, \ldots, k$ we obtain an element x of F_q corresponding to $mk + j$. For such an x, we compute the right side of the equation

$$y^2 = f(x) = x^3 + ax + b,$$

and find a quadratic residue modulo q.

In practice $k = 30$ or at worse $k = 50$ suffices. In our implementation through repeated trials, it was found that the average number of tries was 3 (i.e., $k = 3$) and the maximum value of k attained was 8.

ElGamal Procedure

Suppose Asha has to send a message P_m to Balu, $P_m \in E(F_q)$. She chooses a random integer κ such that $0 < \kappa < \#E(F_q)$ and sends the following pair of points as the ciphertext to Balu.

$$C = (\kappa P_B, P_m + \kappa(\vartheta_B P_B))$$

Here $\vartheta_B P_B$ is the public key of Balu.
To decrypt the message, Balu multiplies κP_B with his secret key ϑ_B and subtracts $\kappa\vartheta_B P_B$ from the second point in the pair to get the original message.

$$P_m = P_m + \kappa(\vartheta_B P_B) - \vartheta_B(\kappa P_B)$$

It is clear that there is a message expansion by a factor 2. For each message point, two points have to be sent as ciphertext. From the security point of view, same κ should not be used for consecutive blocks of encryption. If κ is used for for more than one block, knowledge of one block P_{m1} of the message enables an intruder to compute other blocks as follows. Let C_1 and C_2 be two consecutive cipher blocks for the messages P_{m1} and P_{m2}. Using the same κ

$$C_1 = (\kappa P_B, P_{m1} + \kappa(\vartheta_B P_B))$$

$$C_2 = (\kappa P_B, P_{m2} + \kappa(\vartheta_B P_B))$$

. Subtracting the second points of C_1 and C_2 we get $P_{m1} - P_{m2}$ and by knowing one the other can be easily calculated. A single elliptic curve $E(F_q)$ defined over a field F_q alongwith the base point P_B can be shared by a group of people. The security of the system depends on the elliptic curve discrete logarithm problem.

3.7.3 ElGamal Scheme over F_{2^m}

Implementation of ElGamal scheme over F_{2^m} ([40], [41]) has a slightly different approach. Let us consider a non-supersingular curve $E : y^2 + xy = x^3 + a_2x^2 + a_6$ defined over F_{2^m}, and let P be a publicly known point on E. Assume that the elements of F_{2^m} are represented in normal basis. Messages are considered as ordered pairs of elements in F_{2^m}. User Asha randomly chooses an integer a and makes public the point aP, and keeps a secret. To transmit the message (M_1, M_2) to Asha, sender Balu selects a random integer κ and computes the points κP and $a\kappa P = (\bar{x}, \bar{y})$. Assuming that $\bar{x}, \bar{y} \neq 0$, since the event $\bar{x} = 0$ or $\bar{y} = 0$ occurs with negligible probability for random κ, Balu then sends Asha the point κP, and the field elements $M_1\bar{x}$ and $M_2\bar{y}$. To read the message, Asha multiplies the point κP by her secret key a to obtain $a\kappa P(\bar{x}, \bar{y})$, from which she can recover M_1 and M_2 in two divisions. By knowing M_1 (or M_2), an intruder can easily obtain M_2(or M_1). This attack can be prevented by only sending $(kP, M_1\bar{x})$.

In this scheme, four field elements are transmitted in order to convey a message consisting of two field elements. There is *message expansion* by a factor of 2. The message expansion can be reduced to 3/2 by sending x_1 and only a single bit of y_1/x_1 (if $x_1 \neq 0$), instead of sending the point $P = (x_1, y_1)$. Then y_1 can be recovered as follows. If $x_1 = 0$, then $y_1 = \sqrt{a_6}$. If $x_1 \neq 0$, then the change of variables $(x, y) \rightarrow (x, xz)$ transforms the equation of the curve to $z^2 + z = x + a_2 + a_6x^{-2}$. Compute $\alpha = x_1 + a_2 + a_6x_1^{-2}$. To solve the quadratic equation $z^2 + z = \alpha$, let $z = (z_0, z_1, \ldots, z_{m-1})$ and $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{m-1})$ be the vector representations of z and α respectively. Then

$$z^2 + z = (z_{m-1} + z_0, z_0 + z_1, \ldots, z_{m-2} + z_{m-1}) \tag{3.8}$$

An unique solution $\bar{z}$ to $z^2 + z = \alpha$ can be obtained by choosing $z_0 = 0$ or $z_0 = 1$. This can be obtained by comparing the components of $z^2 + z$ and α. The exact solution can be obtained by comparing it with the corresponding bit of y_1/x_1 that was transmitted. y_1 is obtained from $y_1 = x_1\bar{z}$.

For the implementation of this Scheme a throughput of 27 Kbps has been achieved using the processor discussed in 3.6.3.

3.7.4 Elliptic Curve analog of RSA

For some special classes of elliptic curves the order and group structure are easy to compute as given by the following lemmas [35].

LEMMA 1 *Let p be an odd prime satisfying $p \equiv 2 \bmod 3$. Then, for $0 < b < p$, $E_p(0, b)$ is a cyclic group of order $\#E_p(0, b) = p + 1$.*

LEMMA 2 *Let p be a prime satisfying $p \equiv 3 \bmod 4$. Then, for $0 < a < p$, we have $\#E_p(a,0) = p+1$. Moreover, $E_p(a,0)$ is cyclic if a is a quadratic residue modulo p and $E_p(a,0) \cong Z_{(p+1)/2} \times Z_2$ otherwise.*

In this section, we look into the implementation of RSA based on elliptic curves over a ring. We describe here a protocol [35] for RSA public key cryptosystem based on elliptic curves as described in Lemma 1. Construction of a system using Lemma 2 is very similar and is not being considered here.

Key Generation

User U chooses large primes p and q such that

$$p \equiv q \equiv 2 \bmod 3$$

U computes the product

$$n = pq, \text{ and } N_n = \text{ lcm } (\#E_p(0,b), \#E_q(0,b) = lcm(p+1, q+1).$$

U then chooses an integer e which is coprime to N_n, and computes an integer d such that

$$ed \equiv 1 (\bmod\ N_n).$$

Now, the secret key of U is d, $(p, q, \#E_p(0,b), \#E_q(0,b), N_n)$, and the public key is n, e.

Encryption

A plaintext $M = (m_x, m_y)$ is an integer pair, where $m_x, m_y \in Z_n$. Let $M = (m_x, m_y)$ be a point on the elliptic curve $E_n(0,b)$, where b is determined by m_x and m_y.

Asha encrypts the point M by encryption function $E()$ with Balu's public key e and n as

$$C = E(M) = e \cdot M \text{ over } E_n(0,b),$$

and sends a ciphertext pair $C = (c_x, c_y)$ to Balu.

Decryption

Balu decrypts the point C by decryption function $D()$ with his secret key d and public key n as

$$M = D(C) = d \cdot C \text{ over } E_n(0,b)$$

In this scheme, message imbedding is not a problem as we take two successive blocks as coordinates of a point and compute b for this point, which decides the curve. This makes this scheme interesting as it is not defined on a single group but on a large class of groups, all with the same order. The curve to be used is thus

determined by the plaintext to be transmitted. For the class of elliptic curves used in this scheme, the addition formula is independent of a and b, and the doubling formula is independent of b. Thus, the above protocol does not require computation of the value $b = y^2 - x^3 \bmod n$. If Lemma 2 is adopted, for the addition formula the sender must compute a such that $a = (m_y^2 - m_x^3)/m_x \bmod n$, and the receiver must compute a such that $a = (c_y^2 - c_x^3)/c_x \bmod n$.

It is to be noted that in the case of Lemma 1, the minimum possible value of e is 5 because $2|N_n$ and $3|N_n$. In the case of Lemma 2, the minimum possible value of e is 3 because $2|N_n$.

Security

Security of this scheme defined over elliptic curves, like conventional RSA, depends on the difficulty of factoring n. Considering the fact that elliptic curve computations are difficult, it is worth to probe the need for RSA over elliptic curves. In broadcast applications, the original RSA cryptosystem is not secure if the encryption key e is small. In other words, an attack based on the Hastad theorem called the *low exponent attack* is effective against this cryptosystem. It has been shown in [kuro, kuwa] that RSA-type cryptosystems over elliptic curves such as the KMOV and Demytko cryptosystems [REF: 3,4], are more secure than the original RSA cryptosystem against the low exponent attack (referred to as *low multiplier attack*, in case of elliptic curves).

Low Exponent Attack ([36], [25], [37])

Theorem (Hastad) 1 *Let e and n_i be public key of the original RSA cryptosystem for a receiver $R_i (1 \leq i \leq k)$. The common plaintext m is encrypted as $c_i = m^e \bmod n_i, (1 \leq i \leq k)$ for k receivers. If $k \geq e$ then the system of congurences $c_i \equiv m^e (\bmod\ n_i), (1 \leq i \leq k)$ can be transformed into the equation $c = m^e$, where c is the combined cyphertext from $c_i, (1 \leq i \leq e)$ via the Chinese remainder theorem.*

Assume that 3 is chosen as the exponent and that A wants to send the same message m to users U_1, U_2 and U_3. She will compute and send $y_i \equiv m^3 (\bmod\ n_i), i = 1, 2, 3$. But using the fact that n_1, n_2 and n_3 are relatively prime a listener who knows the values of y_1, y_2 and y_3 can combine the messages by chinese remaindering to get $m^3 (\bmod\ n_1 n_2 n_3)$ and since $m^3 < n_1 n_2 n_3$ he can recover m. In general if the exponent is e the number of messages needed is e. Instead of sending the same message m to everybody one can attach the time-stamp or user-ID and thus send the encryption of $2^{|t|}m + t$ where $2^{|t|}m$ is the shifted message and t is the time (which will be different for different receivers). The previous attack fails and now the problem is (for $e = 3$) transformed as follows.

Given $(a_i m + b_i)^3 (\bmod\ n_i)$ where all the a_i and b_i are known, recovering m in polynomial time. Hastad proved that it is possible through the following theorem.

Theorem (Hastad) 2 *Given a set of equations* $\sum_{j=0}^{h} a_{ij}x^j = 0 \bmod n_i, i = 1, \ldots, k$ *where* $x < n$ *and* $\gcd(\langle a_{ij}\rangle_{j=0}^{h}, n_i) = 1$ *for all* i. *Then it is possible to recover* x *in polynomial time in* e,k *and* $\log n_i$ *if*

$$N > n^{\frac{h(h+1)}{2}}(k+h+1)^{\frac{k+h+1}{2}}2^{\frac{(k+h+1)^2}{2}}(h+1)^{h+1}$$

where $N = \prod_{i=1}^{k} n_i, n = \min(n_i), h$ *is the degree of the equation and* k *is the number of equations.*

Therefore, sending more than $\binom{e(e+1)}{2}$ messages enables an adversary to recover the messages and sending linearly related messages using conventional RSA with low exponent is also insecure.

However, this attack is not known against elliptic curve RSA. The security of elliptic curve RSA cryptosystems has been evaluated in ([37] and [36]) against the Hastad attack. Kuwakado [36] showed that if $e \geq 5$ and $n = 2^{511}$, then elliptic RSA cryptosystems are secure against the Hastad attack. It is to be noted here from lemma 1 the minimum possible value for e is 5. For this value of e, for k = 428, Kurosawa [37] showed that these schemes are not secure if $n \geq 2^{1024}$. For values of $n < 2^{1024}$ the condition in the theorem above is not satisfied and hence are secure.

Chapter 4

Multiple Precision Arithmetic

4.1 Introduction

Given the background in the previous chapters, let us now look into the implementational aspects of a cryptosystem. It is clear from the earlier discussions that the parameters involved in various computations are *BIG NUMBERS.* For example consider working in a field $F_{2^{155}}$. This would mean that all the computations are with 155-bit numbers. Similarly, if we are working in a field F_p where p is a 100 digit prime, then the computations involve 100 digit numbers. Today to implement a secure RSA system, we must be able to manipulate 512-bit numbers.

To handle numbers of this size we need a tool, often referred to as *"multiple precision integer arithmetic"* (MPIA) package, which can manipulate numbers of large precisions. In general, there are several implementations of MPIA that are available. The first significant implementation was by Buell and Ward [19]. They developed a package for MPIA and number-theoretic computation on the CRAY-2 and it was written entirely in FORTRAN. RSA Laboratories, which holds the patent for the RSA Cryptosystem, has its own *MP* arithmetic cryptographic tool-kit, called RSAREF. This is accessible to only US and Canadian citizens and has export restrictions even for non-commercial applications. Also, there are other implementations specific to certain processors. Significant among these is an implementation on Motorola DSP56000 [20], highly optimized for that processor. Apart from these there are multiprecision libraries like *mplib* and *gmplib* in UNIX environment. These libraries have the support for just the basic arithmetic and doesn't include any number-theoretic and cryptographic functions.

The growing importance of data security, combined with the increased power and omnipresence of PCs, emphasise the need for cryptography on the desktop. We need an implementation of MPIA that can run on the ubiquitous PCs. We now discuss about how we operate on multiprecision (MP) numbers using a programming language such as *C* that only goes as far as 32 bits.

4.2 Basic Arithmetic with MultiPrecision Numbers (MPs)

4.2.1 Representation of MultiPrecision Numbers (MPs)

MPs are represented ([13], [12]) as arrays of type MP_DIGIT, where MP_DIGIT depends on the machine. For a 32-bit machine, MP_DIGIT can be defined as an unsigned long, which has a size of 32 bits. Each element of the array is a digit in the base r representation of the MP, where $r = 2^b$. For a 32-bit machine we can have $b = 32$. The representation of an integer x as an n-digit array is shown by the summation below.

$$x = \sum_{i=0}^{n-1} x[i]r^i$$

The minimum value of a digit is 0 and the maximum is $r - 1$. Lower-indexed elements of the array are less significant than higher-indexed elements. We can therefore define a multiprecision integer to be of type MP_INT which is an array of MP_DIGITs. For an MP_INT a, $a[0]$ is the 1s digit of the array, a[1] the rs digit (similar to 10s digit), a[2] the r^2s digit (similar to 100s digit), and so on.

For example, $2^{512}+1$ (ninth Fermat number) , would be represented as an array of 17, 32-bit MP_DIGITs:

$$a[0] = 1$$
$$a[1] = a[2] = \ldots = a[14] = a[15] = 0$$
$$a[16] = 1$$

With C's built in addition, subtraction, multiplication, and division operators, we already have the following abilities.

- Add two MP_DIGITs, and get the 1s digit of the sum (but not the carry-out).
- Subtract an MP_DIGIT from an MP_DIGIT, and get the 1s digit of the remainder (but not the borrow-out).
- Multiply two MP_DIGITs, and get the 1s digit of the product (but not the rs digit).
- Divide an MP_DIGIT by an MP_DIGIT, and get the quotient, also an MP_DIGIT.

We have to now build the ability to operate on MP_INT using the above. Adding and subtracting MPs is quite easy; multiplying them is harder; and dividing is the hardest. In this chapter we wouldnot be discussing normal integer division as division in cryptographic implementations are very specific. The computations involved

are always in modular arithmetic where all results are divided by a predetermined quantity called the "modulus", and only the remainder is kept.

Later in this chapter we would discuss a special implementation of modular arithmetic due to Montgomery [45]. This method avoids division in modular reduction. Modular exponentiation in particular, are essential and time-critical part of the RSA and DSS schemes. We discuss efficient methods for implementing this in the next chapter.

4.2.2 MP Addition

Adding two MPs is much the same as classroom method. Given a carry-in that's either 0 or 1 and two addend digits (MP_DIGIT), we have to compute the sum digit and a carry-out. Since we represent each digit by 32 bits, it's difficult to get the carry-out, but can be accomplished by applying a twist as follows.

- Add the carry-in to the first addend digit. Let us denote this sum as "subsum".
- If the subsum is less than the carry-in, then there is a carry-out. This is because the real sum has wrapped past the maximum digit and it cannot get as far as the carry-in. this implies that the carry-in is 1, and the subsum is $0(<$ carry-in). So, write down the second addend digit as the sum digit, and go on to the next digit.
- If subsum $\geq$ carry-in in the previous step, add the subsum to the second addend digit. If the sum digit is less than the second addend digit, we have to carry out and otherwise not.

Example:

Let us add 4532 to 8097.

```
 1 0 1 0 0   ←  carry
   4 5 3 2   ←  first addend
   4 6 3 2   ←  subsum
   8 0 9 7   ←  second addend

 1 2 6 2 9   ←  sum
```

4.2.3 MP Subtraction

Subtracting two MPs is just like adding two MPs, except that we borrow instead of carry. Given a borrow-in that's either 0 or 1, a subtrahend digit, and a minuend digit we have to compute a remainder digit and a borrow-out.

- Subtract the borrow-in from the subtrahend digit. Let this be the "subremainder".
- If it is more than the maximum digit minus the borrow-in, we have to borrow out which means that the borrow-in is 1, and the subremainder is the maximum digit. Since the subremainder is the maximum digit, we write down the maximum digit minus the minuend digit as the remainder digit and go on to the next digit.
- If there is no borrow-out in the previous step, subtract the minuend digit from the subremainder which we write down as the remainder digit. If the remainder digit is more than the maximum digit minus the minuend digit, we have to borrow out and otherwise not.

4.2.4 MP Multiplication

Let us compute $a = bc$, where b and c have n digits, and a has $2n$ digits.

Operand Scanning method

The normal classroom approach to this multiplication can be written down as given by the expression below.

$$a = \sum_{i=0}^{n-1} b[i] r^i c$$

This method is referred to as multiplication by "operand scanning". Here we multiply by digits of the operand b from least to most significant, following the weights r^i.

Algorithm for operand scanning is shown below.

```
a ← 0
for i ← 0 to n − 1
  do x ← 0
    for j ← 0 to n − 1
      do x ← x + a[i+j] + b[i]c[j]
        a[i + j] ← x mod r
        x ← ⌊x/r⌋
    a[i + n] ← x
```

In this method we compute the product by accumulating partial products $b[i]r^i c$ for each i. There are n iterations. The variable x carries between iterations of the j loop. There is no carry between iterations of the i loop. Given below is an example

to illustrate this method.

```
          5 4 3 2   ←  multiplicand
          9 8 7 6   ←  multiplier
          0 0 0 0   ←  accumulator value
        3 2 5 9 2   ←  intermediate product
        3 2 5 9 2   ←  accumulator value
      3 8 0 2 4     ←  intermediate product
       4 1 2 8 3 2
     4 3 4 5 6
     4 7 5 8 4 3 2
   4 8 8 0

   5 3 6 4 6 4 3 2   ←  final product
```

Product Scanning Method

The expression below gives a different approach that leads to multiplication by "product scanning," which is similar to convolution in signal processing.

$$a = \sum_{k=0}^{2n-1} r^k \sum_{i=\max(0,k-n+1)}^{\min(k,n-1)} b[i]c[k-i]$$

Algorithm for product scanning is given below.

$x \leftarrow 0$
`for` $k \leftarrow 0$ `to` $2n-1$
  `do for i` $\leftarrow \max(0, k-n+1)$ `to` $\min(k, n-1)$
    `do` $x \leftarrow x + b[i]c[k-i]$
  $a[k] \leftarrow x$
  $x \leftarrow \lfloor x/r \rfloor$

We compute digits of the product a from least to most significant, following the weights r^k. Given below is an example for multiplication by product scanning. We compute the cross products between pairs of operand digits and their columnwise sum, with carry propagation right to left, gives the product.

			5	4	3	2	←	multiplicand
			9	8	7	6	←	multiplier
8	10	10	6	3	1		←	carry
			18	16	14	12		
		27	24	21	18			
	36	32	28	24				
45	40	35	30					
3	6	4	6	4	3	2	←	product

It's clear that in operand scanning, there are n^2 iterations of the j loop, plus whatever overhead there is in the n iterations of the i loop. Product scanning, on the other hand, computes the product by accumulating partial products for each k.

$$r^k \sum_{i=\max(0,k-n+1)}^{\min(k,n-1)} b[i]c[k-i]$$

There are $2n$ iterations. The variable x accumulates within an iteration of the i loop, and between iterations of the k loop. Its value is always less than nr^2. If $n \leq r$ then x needs at most three digits.

The index $k - i$, like i, is always between 0 and $n - 1$. On the last iteration of k loop, i ranges from n to $n - 1$, so the i loop has no iterations. There is an overhead in the $2n$ iterations of the k loop. This method is simpler than the operand-scanning method and has been proved ([13], [20]) to be 25 percent faster per iteration. Although there are more iterations of the k loop here than iterations of the i loop in the operand-scanning method, the additional overhead is not significant compared to the savings.

Comparison

Product scanning stores each product digit once, not after every multiply. Of course, it fetches operand digits before every multiply. But it does not fetch product digits at all! In all, product scanning has about one-third fewer memory references than operand scanning. It also has many fewer "shifts".

Product scanning needs more register storage than register scanning. The variable x in product scanning needs at least three digits, whereas in the operand-scanning method needs only one digit.

4.3 Modular Arithmetic

4.3.1 introduction

Modular reduction is a basic operation in many of the cryptosystems. Computation of the form $(A * B)$ *mod* N is required extensively and repetitively. Hence, even a small improvement or optimisation in Modular arithmetic significantly improves the throughput of the cryptosystems. Modular multiplication is half multiplication and half division. Results are divided by a predetermined quantity (modulus) and only the remainder is used. Division takes longer time and is difficult to implement. Montgomery multiplication is a good alternative. It can be implemented by both product and operand scanning.

4.3.2 Montgomery's method

This method[1][45] uses a novel approach to modular reduction without requiring any division operation. Since division is costly to perform, this method is advantageous. However in this method, initially, the problem variables are to be transformed into a special N-residue form. To reduce any integer *mod* N, we define an alternate representation of the integers $(0, 1, 2, ..., N-1)$ called montgomery representation. We translate normal integers to this new form, do our multiplications and then finally translate back the result to the normal representation.

To form the montgomery representation, we choose some R such that $R > N$ and relatively prime to N ($gcd(R, N) = 1$). By choosing R to be some power of 2, division can be made inexpensive since division by any power of two means just right shifts. This also means that N is to be odd as R and N are to be co-primes. This is not a constraint since, in cryptosystems, particularly RSA, N is a product of large primes which assures that N is odd. Generally R is chosen to be a power of the machine word size.

Now to represent the integers in $\{0, ..., N-1\}$, we use the Montgomery representation

$$M(x) = xR \ mod \ N$$

To convert from Montgomery representation to normal representation, we find the inverse of R *mod* N under multiplication *mod* N. Let it be R' . Then the inverse of $M(x)$ is

$$M'(x) = xR' \ mod \ N$$

Since R and N are relatively prime, these functions are one-to-one on $\{0, 1, 2, ..., N-1\}$. Hence all these integers have a unique representation.

[1] "$x = y \ mod \ n$" implies that x is the least residue of y *mod* n, and "$x == y \ mod \ n$" means that x is congruent mod n to y.

Now we find N' such that

$$0 < N' < R \quad and \quad RR' - NN' = 1$$

That is R' ($0 < R' <$ N) is the inverse of R under multiplication *mod* N, and $N - N'$ is the inverse of N under multiplication *mod* R. Either R' or N' can be computed using extended euclidean algorithm and the other from the relation $RR' - NN' = 1$. Due to [20] we have a better algorithm to compute the inverse of x *modulo* y for the special case when x is odd and y is a power of 2. Since the computation of N' is same as this special case, we first compute N' and then R' is obtained from the relation $RR' - NN' = 1$.

function Modular_Inverse(N,R)

```
begin
y[1] := 1
for i := 2 to log2(R) do
if N * y[i - 1] < 2^(i-1) mod 2^i
then y[i] := y[i - 1]
else y[i] := y[i - 1] + 2^(i-1)
return y[i]
end
```

We now define the procedure REDC(x) which has two uses. The first is to compute $M'(x)$. That is, it computes the normal representation of an integer, given the Montgomery representation. The other is to multiply two numbers modulo N in Montgomery representation with the product in Montgomery form.

function REDC(x)

```
begin
N' := -N^-1 mod R  (1)
m := ((x mod R) * N') mod R  (2)
t := (x + m * N)/R  (3)
if t < N  (4)
return t  (5)
else return t - N  (6)
end
```

It is to be noted that in step 3, $x + mN$ has to be divisible by R.

$$mN = ((x \bmod R)N' \bmod R)N == xNN' \bmod R == -x \bmod R$$

$$x + mN == x + (-x) \bmod R == 0 \bmod R$$

Thus R divides $x + mN$. Also, $x + mN == x \ mod \ N$ (since it is x plus a multiple of N), so $tR == x \ mod \ N$ (since $t = (x + mN)/R$) and $t == xR' \ mod \ N$ (multiplying both sides by R'). Thus t is congruent $mod \ N$ to the desired result. Since

$$m < R$$

$$x + mN < RN + RN$$

$$(x + mN)/R < 2N$$

it implies that if $x < RN$, then $t < 2N$, so either t or $t - N$ is the result. Thus REDC(x) returns $xR' \ mod \ N$ given $0 \leq x < RN$. The procedure REDC divides only by R, so all divisions can be done by shifting out low order bits. The time consuming part of REDC is the two multiplications.

We will now look at multiplying two integers which are in Montgomery representation with the product in Montgomery representation. Suppose the integers are x and y, then their Montgomery representations are $xR \ mod \ N$, and $yR \ mod \ N$. Now given $xR \ mod \ N$ and $yR \ mod \ N$, we have to compute $xyR \ mod \ N$. Since R' is the modulo N inverse of R,

$$xyR \ mod \ N = xRyRR' \ mod \ N = (xR \ mod \ N)(yR \ mod \ N) * R' \ mod \ N$$

So if we take the ordinary product of the representations,$(xR \ mod \ N)(yR \ mod \ N)$, and run it through REDC, we get

REDC($(xR \ mod \ N)(yR \ mod \ N)$) $= xyR \ mod \ N$.

Since $(xR \ mod \ N)$ and $(yR \ mod \ N)$ are both less than N, their product is less than NN which is less than RN, so it forms a valid input for REDC. Thus to multiply modulo N, two numbers in Montgomery representation with the product in Montgomery form,

```
function MontyMult(x,y)
    begin
    return REDC (x * y)
    end
```

4.3.3 Multiprecision case

In actual applications, the parameters N, R and the input X are multiple-precision integers and hence involve multiprecision arithmetic. That is, an integer A is represented as a sequence of digits $a_0, ..., a_{(n-1)}$ where

$A = a_{(n-1)}b_{(n-1)} + a_{(n-2)}b_{(n-2)} + ... + a_1b + a_0$ and b is the base, typically a power of 2 (word size of the processor) and n is the number of digits. Also we choose

$R = b^n$. In the function REDC, step-2, instead of computing all of m at once, we can do it one digit m_i at a time. This allows us to use a single digit n'_0, which is $-n_0^{-1}$ *mod* b instead of $-N^{-1}$ *mod* R.

function REDC(x)

```
begin
n'_0 := -n_0^{-1} mod b
t := x
for i := 0 to n - 1 do
begin
m_i := x_i * n0' mod b
t := t + m_i * N * b^i
end
return t/R
end
```

Similarly, montgomery product of two numbers can be computed using the algorithm below.

function MontyMult(A,B)

```
begin
n'_0 := -n_0^{-1} mod b
T := 0
for i := 2 to n - 1 do
begin
T := T + A_i * B * b^i
m_i := T_i * n'_0 mod b
T := T + m_i * N * b^i
end
return T/R
end
```

Let us denote the montgomery reduction modulo N as $M_N()$. Following observations are worth mentioning:

- It involves only multiplication. The operation *mod* b^n is just truncation, since b is the digit radix.
- If x and y are between 0 and $N-1$, then $M_N(x,y)$ is between 0 and $2N-1$ and can be reduced modulo N with, at most, one subtraction.
- It obeys the ususal multiplicative laws: $M_N(x,y) = M_N(y,x)$; and $M_N(M_N(x,y),z)$ $M_N(x,M_N(y,z))$. The identity is R *mod* N.

- It relates to ordinary modular multiplication through the ratio $R \ mod \ N$. $M_N((xR \ mod \ N), (yR \ mod \ N)) = xyR \ mod \ N$. This implies that the ordinary product of montgomery representations is the montgomery representation of ordinary product.

Thus Montgomery multiplication lets us avoid a division at the expense of two multiplications. Of course we have the overhead of the conversion to and from Montgomery representation. Typically, when exponentiating mod N, intermediate results are not translated back to normal representation between multiplications; they are left in Montgomery representation until all the multiplications are done. So it is a good trade-off considering the large number of reductions to be done before converting the result.

Chapter 5

Implementation and Results

5.1 Introduction

The calculation of modular products are an important part of software implementations of many cryptographic protocols such as the well-known RSA Public Key algorithm. In these exponential cryptographic systems, there is a need for fast modular exponentiation, that is the calculation of

$$C = M^e \bmod n$$

where for acceptable levels of security C, M, e, and n are multiprecision numbers. Similarly, the basic operation performed on an elliptic curve cryptosystem is the computation of multiplicity $d \cdot P$ of a point P on the elliptic curve modulo n. This corresponds to the computation of $M^e \bmod n$. Multiplication and squaring in the latter is replaced by addition and doubling in the former. For a large n and d (and e), the time complexity of elementary operations as well as the number of elementary operation are very high. Thus, reducing the number of such operations is important when implementing the above algorithms. By combining ideas from [28] and Dusse and Kaliski [20], and exploiting Montgomery's method [45] for modular reduction, efficient routines have been developed.

Let us first consider the computation $C = M^e \bmod n$ The conventionally used algorithm for this is the binary method, described by Knuth [30].
The algorithm is as shown below.

$C \leftarrow M$
for $i \leftarrow \lambda(e) - 1$ to 0 step -1
  do $C \leftarrow C^2 \bmod n$
    if $e_j = 1$
      then $C \leftarrow CM \bmod n$

The above problem can conveniently be divided into two phases; the squaring or multiplication of two large multiprecision numbers, and then their subsequent reduction to a remainder when divided by n . This reduces the above calculation to a sequence of squarings and multiplications, modulo n , the total number of which depends on $\nu(e)$, the number of 1s (hamming weight) in the binary representation of the exponent. This method takes $\lambda(e)+\nu(e)$ modular multiplications, where $\lambda(e)+1$ is the bit length of the exponent. Let $\langle e_{\lambda(e)}, \ldots, e_0 \rangle$ be the bit representation of e. In this method, for each bit of the exponent except the first, squaring is done and if the bit is 1, then a multiplication is also done.

Therefore fast modular exponentiation can be achieved if we have access to fast methods for modular squaring and multiplication and also if the hamming weight of the exponent can be reduced.

In case of elliptic curves, the m-ary algorithm for computing $R = d \cdot P$ can be given as below.

$R \leftarrow P$
for $i \leftarrow \lambda(d) - 1$ to 0 step -1
  do $R \leftarrow 2R$
    if $d_j = 1$
      then $C \leftarrow P + R$
where P and R are points on an elliptic curve.

So, it is clear that given a method for reducing the hamming weight of d (or e), we can reduce the number of elementary operations in the computation of modular exponentiation and multiplicity of points. In this implementation Montgomery's modular reduction technique has been used for squaring and multiplication. As discussed in the previous chapter, this gives us an efficient implementation of multiplication and squaring. For exponent reduction, signed binary window method [28] has been implemented. This algorithm is applicable to the computation of both modular exponentiation and multiplicity of a point on an elliptic curve as will be shown in the next section.

To prove the efficiency of Montgomery's algorithm, timing tests were conducted and the results have been shown in Table 5.1. For each size of the modulus, the modulus and the exponent of that size was fixed and random numbers were generated for the base. For each random number, computations were done with montgomery reduction and normal reduction method. The timings shown are the average of several runs for a particular modulus size.

Table 5.1: Average computation time for Montgomery reduction in a PC-AT 486-DX2, 66MHz

Size of the Modulus	Normal method	Montgomery's method
128 bit	6 sec.	.1 sec.
256 bit	9 sec.	.5sec.
512 bit	14 sec.	3.55 sec.
1024 bit	26 sec.	5.33 sec.

5.2 Signed Binary Window Method

The computation of the kind M^e mod n is similar to the basic operation performed on an elliptic curve which is the computation of a multiple d.P of a point P on the elliptic curve modulo n, For a large n and d. the time complexity of elementary operations as well as the number of elementary operation are very high. Thus. reducing the number of such operations is important when implementing cryptosystems which are time critical.

In this section, we describe the signed binary window method proposed by Koyama and Tsuruoka [28]. This method is based on pre-computation to generate an adequate addition-subtraction chain for multiplier, the d. In this method the multiplier d is not represented in binary but in a special form called signed binary representation. This increases the average length of zero runs in a signed binary representation of d, and speeds up the binary window method. Eventhough this discussion is specific about computation on elliptic curves, all the results are directly applicable for modular exponentiation and will be shown at the end of this section.

An addition chain for a given d is a sequence of positive integers

$$a_0(=1) \rightarrow a_1 \rightarrow a_2 \rightarrow \ldots \rightarrow a_r(=d),$$

where r is the number of additions, and $a_i = a_j + a_k$, for some $k \leq j < i$, for all $i = 1, 2, \ldots, r$. To evaluate d.P or x^d, the ordinary binary method without pre-computation requires $(3/2)[\log_2 d]$ multiplications on average. The ordinary binary method does not always guarantee the minimum number of multiplications (the shortest addition chain). Obtaining the shortest addition chain is a NP-complete problem.

An addition chain can be extended to an addition-subtraction chain, with a rule $a_i = a_j \pm a_k$ in place of $a_i = a_j + a_k$. This idea corresponds to the evauation of x^d using multiplication and division. For integers, division (or the computation of a multiplicative inverse modulo n) is a costly operation, and implementing this idea does not seem feasible. In case of elliptic curves the division in Z_n is replaced by a subtraction, which has the same cost as an addition. An addition (subtraction) formula on elliptic curves does not contain a division in Z_n particularly when homogeneous coordinates are used. Thus, the addition-subtraction chain can be effectively applied to computations over elliptic curves.

The Koyama-Tsuruoka method is a window method and the multiplier d is represented in signed binary form. For a given number d, this method consists of the following four phases:

- Representation of d
- Splitting the representation into segments (windows),
- Computing the segments,
- Concatenating the segments.

Representation

For a given number d, this method uses the signed (three-valued) binary representation $T : [t_{L-1}, \ldots, t_1, t_0], t_{L-1} \neq 0$ for d satisfying $d = \sum_{i=0}^{L-1} t_i 2^i$, where $t_i \in \{\bar{1}, 0, 1\}$, and $\bar{1}$ denotes -1. It is to be noted that in ordinary binary representation B is uniquely determined for a given d, but T is not unique. In this method a transform algorithm is used to transform B to T such that it increases the average length of zero runs and minimizes the weight of T. The average length of the zero runs in T, denoted by $Z(T)$, is defined as follows.

$$Z(T) = \frac{1}{L} \sum_{i=0}^{L-1} z(i),$$

$$z(i) = \begin{cases} 1 + z(i-1) & if\, t_i = 0 \\ 0 & if\, t_i \neq 0 \end{cases} (0 \leq i \leq L-1),$$

where $z(-1) = 0$.
Let B' be a subsequence of B, and let T' be a subsequence of T. A rule for transforming B' to equivalent T' is as follows.

Transformation Rule

$B' : (1 \ldots b_i \ldots)$ can be transformed into $T' : [10 \ldots . t_i \ldots \bar{1}]$, where $t_i = b_i - 1$.

Let $\#_0(B')$ be a number of zeroes in B', and let $\#_1(B')$ be a number of non-zero digits in B'. The weight of T' is estimated as $\#_1(T') = 2 + \sum |t_i| = 2 + \sum |b_i - 1| = 2 + \#_0(B')$. Thus the weight decreases by the transformation if $\#_1(B') - \#_0(B') > 2$.

The transform algorithm inputs B in LSB first order and counts the difference $D(B') \equiv \#_1(B') - \#_0(B')$, and applies the transformation rule repeatedly to appropriate B' with $D(B') \geq 3$. The output of this method is not sparse. The output is said to be sparse if no two adjacent digits are nonzero.

The transform algorithm is given below.
algorithm *transform* (input B: array, output T: array)

```
begin
  M := 0; J := 0; Y := 0; X := 0; U := 0; V := 0; W := 0; Z := 0
  while X < ⌊log2 d⌋ do begin
    if B[X] = 1 then Y := Y + 1 else Y := Y − 1;
    X := X + 1;
    if M = 0 then begin
      if Y − Z ≥ 3 then begin
        while J < W do begin T[J] := B[J]; J := J + 1 end;
        T[J] := −1; J := J + 1; V := Y; U := X; M := 1
      end else if Y < Z then begin Z := Y; W := X end
    end else begin
      if V − Y ≥ 3 then begin
        while J < U do begin T[J] := B[J] − 1; J := J + 1 end;
        T[J] := 1; J := J + 1; Z := Y; W := X; M := 0
      end else if Y > V then V := Y; U := X end
    end
  end;
  if M = 0 ∨ (M = 1 ∧ V ≥ Y) then begin
    while J < X do begin T[J] := B[J] − M; J := J + 1 end;
    T[J] := 1 − M; T[J + 1] := M
  end else begin
    while J < U do begin T[J] := B[J] − 1; J := J + 1 end;
    T[J] := 1; J := J+1;
    while J < X do begin T[J] := B[J]; J := J + 1 end;
    T[J] := 1; T[J+1] := 0;
  end
  return T
end
```

Splitting

Let w be the width of the window. T is split into segments with a length at most w. The following splitting procedure generates a list of all segments. The input array is represented by T.

procedure split (input T: array, w: inter, output S: array)
  Let segment list S be empty
  while (length(T) $\geq w$)
   begin
    Let W be the left w digits of T.
    Let R be T excluding W.
    Let $\tilde{W}$ be W excluding the right 0's.
    Let $\tilde{R}$ be R excluding the left 0's.
    Add new segment $\tilde{W}$ to segment list S
    T:= $\tilde{\text{R}}$.
   end
  Add last segment T to segment list S
  return S

Computing the Segments

In B, the value of each segment is an odd positive integer upto $2^w - 1$. In T, if $w \geq 3$, the segment value never becomes $2^w - 1$ or $-(2^w - 1)$ because of the property of the transform algorithm. Each segment value is an odd integer from $-(2^w - 3)$ to $2^w - 3$. The absolute values of all segments are obtained by the following simple addition sequence. (i.e. $1, 2, 3, 5, 7, \ldots, 2^w - 3$)

$$a_0 = 1, a_1 = 2, a_2 = 3, a_i = a_{i-1} + 2$$
$$(3 \leq i \leq 2^{w-1} - 1)$$

Therefore, in T, all segment values can be computed by at most 2^{w-1} additions.

Concatenating and the Number of Operations

Concatenation requires doublings and non-doubling additions. The segment values are concatenated as follows:

$$dP = ((\ldots((d_kP \cdot 2^{z_k+L_{k-1}} + d_{k-1}P) \cdot 2^{z_{k-1}+L_{k-2}} + d_{k-2}P \ldots) \cdot 2^{z_2+L_1} + d_1P) \cdot 2^{z_1}$$

The inner most D_kP corresponds to the most significant segment, and the exponent $z_k + L_{k+1}$ corresponds to the sum of the length z_k of the following window gap and the length L_{k+1} of the next segment.

Let L be the length of B or T. Note that L is $\lambda+1$ for B and L is $\lambda+1$ or $\lambda+2$ for T. Let Z' be the average length of zero runs in the most significant windows for B or T. In other words, Z' is the average number of $0's$ deleted in W by the splitting algorithm in the beginning. Let Z'' be the average length of zero runs deleted in R by the splitting algorithm for B or T. The average length of the most significant segment is $w - Z'$. The number of doublings in concatenation is same as the length of T(or B) except for the most significant segment. Thus, the number of doublings in concatenation is $L - (w - Z')$ for B and T. The average number of segments becomes $L/(w + Z'')$, which corresponds to the number of non-doubling additions in concatenation.

Thus, on average, the window method requires R operations:

$$R = (L + Z' - w) + \frac{L}{w + Z''} + C,$$

where $C = 2^{w-1}$ for B, and $C = 2^{w-1} - 1$ for T.

In case of modular exponentiation, where d is the exponent in x^d, it can be computed from the following relation.

$$x^d = ((\ldots((x^{d_k})^{2^{z_k+L_{k-1}}} \cdot x^{d_{k-1}})^{2^{z_{k-1}+L_{k-2}}} \cdot x^{d_{k-2}} \ldots)^{2^{z_2+L_1}} \cdot x^{d_1})^{2^{z_1}}$$

5.2.1 Implementation

The routines outlined above have been implemented using C. In order to demonstrate the efficiency of SBW method for modular exponentiation, the conventional RSA algorithm has been implemented.

Average time for, RSA computation for bit-size 128, 256, 512 and computation of multiplicity of a point on an elliptic curve are shown in the Tables 5.2 and 5.3 respectively for Knuth's binary method and signed binary window method These timings are for a PC-AT 486-DX2, running at 66 MHz. Given below is an example for SBW method, generated by the program.

Table 5.2: Average time for the computation of RSA computation in a PC-AT 486-DX2, 66MHz

RSA size	binary method	signed binary method
128 bit	1.2 sec.	.25 sec.
256 bit	4.3 sec.	1.75 sec.
512 bit	10.26 sec.	4.86 sec.

```
-----------------------------------------------------------------
          This is an Implementation of KOYAMA-TSURUOKA SBW Method
-----------------------------------------------------------------

The value of 'd' you have typed in is
436913269784326597843659436
Enter the window size          <4>
Input in binary form is
B: 10110100101100111111110111101000001111 1
   00001110110000111010101101000100001010 1
   01010101100
No of bits = 89

MSD representation is
T: 1 0 -1 0 0 -1 0 -1 -1 0 -1 0 0 -1 -1 0 0 0 0 0
   0 0 0 -1 0 0 0 0 -1 -1 0 0 0 0 1 0 0 0 0 -1 0
   0 0 1 0 0 0 -1 0 -1 0 0 0 1 0 0 0 -1 0 -1 0 -1
   0 0 -1 -1 0 0 0 1 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1
   0 1 1 0 0

The values computed are:
              d_k  : 3 -11 -9 -1 -1 -3 1 -1 1 -5 1 -5 -9 -1 1 5 5 5 3 0
            L_{k-1} : 3 4 4 1 1 2 1 1 1 3 1 3 4 1 1 3 3 3 2 0
            z_k    : 2 1 0 8 4 4 4 3 3 3 1 0 3 4 1 1 1 2 0
```

Table 5.3: Average time for computation of multiplicity of a point on an elliptic curve $(d \cdot P)$

Size of the multiplier d	Double & ADD method	Signed Binary Window method
30 digits	2.0329 sec.	1.7033 sec.
40 digits	2.6373 sec.	2.1978 sec.
55 digits	3.6813 sec.	3.0769 sec.
60 digits	4.0109 sec.	3.2967 sec.

5.3 A Package for Computation on Elliptic Curves

Before starting any implementation over elliptic curves, we need to build a library which can support arithmetic over elliptic curves. We have adapted a software package for computation on elliptic curves. This software facilitates computation using algebraic structures and number theory with emphasis on elliptic curves. The design of the software was influenced by SIMATH, a computer algebra system developed at the Universität des Saarlandes in Saarbrücken (Germany). Some of the structures have been adapted from SIMATH while due care has been taken to optimise the computationally-intensive portions. SIMATH has been designed for UNIX systems and runs on HP-UX and SUN OS. We have successfully implemented this package in a PC which was a challenging task. This package works on any PC with i386 and above, and running MSDOS. The porting has given the package a wider reach.

The package also consists of an interactive calculator which consists of

- many of the system functions
- comprehensive error checking
- detailed "help features".

An overview of the System

The system consists of

- the *basic* system which consists of managing programs, modified input/output functions, and a *list* system with an automatic *garbage collector* and dynamic memory administration;
- a *multiple precision arithmetic* package for computations over $\mathbb{Z}$, $\mathbb{Z}/m\mathbb{Z}$ and finite fields;
- a *polynomial* package for computations with polynomials in any number of unknowns over any of the structures contained in the arithmetic package;
- a *matrix vector* package for matrix/vector computations over the structures contained in the arithmetic package and over polynomial rings;
- an *elliptic curves* package with elliptic-curve-specific functions over the structures contained in the arithmetic package;
- software libraries for user applications;

- the interactive calculator;
- an *User interface*: the link between the user and the system;

The elliptic curve package contains functions for

- Arithmetic over elliptic curves.
- combined Schoof-Shank's algorithm for counting points on elliptic curves over prime fields and finite fields of characteristic 2;
- algorithm for constructing elliptic curves with a given number of points over a given field;
- conversion from affine to projective coordinates and vice versa.

This package has been used for the implementations discussed in the next section.

5.4 Implementation of Elliptic Curve Cryptosystems

5.4.1 ElGamal Scheme

The ElGamal scheme described in Section 3.7.2 has been implemented in software. The software uses three files namely, *publ_key.elg*, *priv_key.elg*, *work_cur.elg*. *publ key.elg* contains the public keys and *priv_key.elg* has the secret key of the user. The file *work cur.elg* contains the parameters of the elliptic curve over which the system works. At the start of the program, the curve parameters are read from this file. By changing the parameters in the curve, the system can be made to work with different set of curves. This makes the system very flexible. Shown below is a sample of the file *work cur.elg*. Considering an elliptic curve $E_p(a, b)$ represented in short normal form, the file shows the parameters p, a, b and the base point P one below the other in that order.

```
19999999999999999999999998058667524308258114418756 9
14848662876205547969080296514941739941090710151604 8
91791942014707744104904176978653306234927171105946
( ( 563944706  652136738  357002604  313126945  393227603  470 )
( 993312320 850608605  520591648  816554363  190865839  65020 )  1 )
```

The software consists of the following five modules.

- Preprocessing
- Encryption
- Decryption
- Key Generation
- Postprocessing

Preprocessing / Postprocessing

The encryption and decryption modules understand only numbers whereas the file to be encrypted consists of text. This module has to be used inorder to convert text to numbers. The plaintext, has to be preprocessed before passing it onto the encryption module. Preprocessing module takes in a text file and outputs a numerical file (file consisting of just numbers). Each character in a file is converted to its ASCII equivalent and is stored in a particular format. Postprocessing module does exactly the opposite. It takes in a numerical file and outputs the text file. The output file from a decryption module is just numbers and has to be postprocessed before recovering the plaintext.

Key Generation

Any new user has to get registered before he can receive any encrypted message. The user chooses an USER-ID and enters a random number. This random number serves as the users secret key k. The program now computes the public key $k * P$ for this user where P is the BASE POINT of the working curve. The points on the elliptic curve are represented as a list of projective coordinates. The public key and the secret key are written in the files publ_key.elg and priv_key.elg respectively. Given below is a sample session for key generation.

```
WORKING CURVE IS EC(a,b) OVER GF(p)       where
a = 148486628762055479690802965149417399410907101516048
b = 917919420147077441049041769786533062349271711059 46
p = 19999999999999999999999998058667524308258114418 7569
The BASE POINT chosen is:
( ( 563944706  652136738  357002604  313126945  393227603  470 )
( 993312320  850608605  520591648  816554363  190865839  65020 )  1 )
press any key to continue
```

```
Enter:
  '1' for encryption
  '2' for decryption
  '3' for Key generation

                                                        <3>
Please enter the USER-ID you require in alphanumeric    <ASHA>
Enter a random number < #EC(a,b)/F_p; This is your secret key.
    <34876583748743687348756865873478563834534535>

The public key 'k*P' of ASHA is :
( ( 103926098  215304201  712251265  111224613  161140750  33503 )
( 758364253 820837412  680992828  253785033  706268666  100574 )  1 )
Key generation successful!!!
press any key to continue
```

A sample of the file *publ_key.elg* is shown below.

```
#GANESH
( ( 412564800  299764211  1042953185  967922145  444487410  71405 )
( 403152028  556419030  38821355  196512697  482737121  19276 )  1 )
#ANJAN
( ( 57688161  367851042  893614798  61834521  635342672  125718 )
( 1010532596 785156749  531332585  83324805  883686276  135944 )  1 )
#UDAYA
( ( 346114033  1021482961  152710560  942706862  486534328  121706 )
( 395376916  281457862  683395352  88474358  259115801  110712 )  1 )
#MADHU
( ( 900766780  108765936  433554337  18198582  406153930  61190 )
( 724791749 359330786  1009904365  200571998  871964376  124938 )  1 )
```

A sample of the file *priv_key.elg* is shown below.

```
#GANESH
984379327498127948723987198373244534
#ANJAN
982758943754536
#UDAYA
5864587346325432074357984798694
#MADHU
876437854990430037490843958390483478
```

Encryption / Decryption

From user point of view encryption module takes-in a preprocessed file, encrypts it and writes the output in the file ?.ELG where ? is the the input filename. Similarly the decryption module takes-in .ELG file and writes the output in the file specified by the user. This file has to be then postprocessed to get back the plaintext. Having explained the algorithm in Section 3.7.2, the procedure is best understood by the example below.

Given below is a sample encryption/decryption session. Anjan is assumed to be sending a secret message to Ganesh. The user response is shown within angles ("< >"). In our examples below we have used the text file "try.txt" to demonstrate the system. Before encrypting, "try.txt" is to be preprocessed. Let the preprocessed file be "try.num". Now, "try.num" is the input file to the encryption module. Similarly, suppose our output file from the decryption module is "try.dec", postprocessing has to be done inorder to obtain the plaintext. Let "try.pla" be the postprocessed file which contains the plaintext.

```
-------------------------------------------------------------------------
  E L - G A M A L     CRYPTOSYSTEM      over  ELLIPTIC CURVES
-------------------------------------------------------------------------


Press any key to continue
                <RETURN>

WORKING CURVE IS EC(a,b) OVER GF(p)       where
a = 14848662876205547969080296514941739941090710151604 8
b = 91791942014707744104904176978653306234927171105946
p = 19999999999999999999999998058667524308258114418756 9
The BASE POINT chosen is:
( ( 563944706  652136738  357002604  313126945  393227603  470 )
( 993312320  850608605  520591648  816554363  190865839  65020 )  1 )
press any key to continue
                                                    <RETURN>


Enter:
  '1' for encryption
  '2' for decryption
  '3' for Key generation

                                                    <1>
```

Table 5.3: Timing details for Elliptic-ElGamal Encryption and Decryption of a file of size 1Kbyte and keysize of 40 digits

MACHINE	ENCRYPTION	DECRYPTION
486DX-33MHz	158 sec.	54 sec.
486DX-66MHz	90 sec.	29 sec.
Pentium-100MHz	44 sec.	14 sec.

```
Enter Your IDENTITY:                                    <ANJAN>
ID check O.K.
Enter Receiver's IDENTITY:                              <GANESH>
ID check O.K.
file to be encrypted :                                  <TRY.NUM>
ENCRYPTING!!!
.........................Encryption process successful!!!
press any key to continue

Enter:
  '1' for encryption
  '2' for decryption
  '3' for Key generation
                                                        <2>
Enter Your IDENTITY:                                    <GANESH>
ID check O.K.
file to be decrypted :                                  <TRY.ELG>
output filename:                                        <TRY.DEC>

DECRYPTING!!!
.........................Decryption process successful!!!
press any key to continue

            <Type any key other than '1', '2' and '3' to exit>
```

Summary

The generated keys are available in the files, "publ_key.elg" and "priv_key.elg". The program reads the details of the curve over which the system works, from the file

Table 5.4: Timing details for Elliptic-RSA Encryption and Decryption of a file of size 1Kbyte and keysize of 40 digits

MACHINE	ENCRYPTION	DECRYPTION
486DX-33MHz	52 sec.	51 sec.
486Dx-66MHz	29 sec.	29 sec.
Pentium-100MHz	14 sec.	14 sec.

"work_cur.elg". To make the system work for a new curve, the file "work_cur.elg" has to be updated. Encrypted output is always available in the file input_filename.elg. For eg: if the input file for encryption is "try.num" then the output is written in "try.elg". eventually "try.elg" would be the input file for the decryption module. The Table 5.3 shows the encryption and decryption time in various PCs.

5.4.2 RSA Scheme

The RSA scheme described in Section 3.7.4 has been implemented in software. The lemma 1 in that section has been used for the implementation. The function and usage of this software is exactly like the ElGamal software except for the algorithms. A summary of the implementation is given below.

Summary

The generated keys are available in the files, "publ_key.rsa" and "priv_key.rsa". Encrypted output is always available in the file input_filename.rsa. For eg: if the input file for encryption is "try.num" then the output is written in "try.rsa". eventually "try.rsa" would be the input file for the decryption module. A point to be noted here is that there is no need for a file like "work_cur.rsa". This is because, the curve depends on the message block and varies with the message block as explained in Section 3.7.4. In other words, elliptic curve analog of RSA doesnot operate on a single curve but on a class of curves all with the same order. The Table 5.4 shows the encryption and decryption time in various PCs. Having described the algorithm in Section 3.7.4, given below are samples of "publ_key.rsa" and "priv_key.rsa". Also shown is an example session of the software.

A sample of the file *publ_key.rsa* is shown below. For each user, n and e are given one below the other.

```
#KUMAR
211264895256552106288097984310241340329
116738042269586220242813478620809002l1
#ASHA
448559004770032616669835444763471
405525059872723057589240507733 29
#BALU
318353212932852963210381802630123702524329027435966023917563091409 9
155855123759236589106632136042964677551141400765642239810797715225
#PANKAJ
211264895256552106288097984310241340329
116738042269586220242813478620809002l1
```

A sample of the file *priv_key.rsa* is shown below. For each user, n and d are given one below the other.

```
#KUMAR
211264895256552106288097984310241340329
178672671869433915259595954398255175 81
#ASHA
448559004770032616669835444763471
633575349284746816067580591859 73
#BALU
318353212932852963210381802630123702524329027435966023917563091409 9
350388442930323509530159590616151146561431047995918918491560928893
#PANKAJ
211264895256552106288097984310241340329
178672671869433915259595954398255175 81
```

An example session for Elliptic RSA encryption followed by decryption:

```
-------------------------------------------------------------------------
     R S A  CRYPTOSYSTEM      over      ELLIPTIC CURVES
-------------------------------------------------------------------------


Enter:
  '1' for encryption
  '2' for decryption
  '3' for Key generation
                                  <1>
Enter Your IDENTITY:              <ASHA>
```

```
ID check O.K.
Enter Receiver's IDENTITY:          <BALU>
ID check O.K.

file to be encrypted :              <try.num>
ENCRYPTING!!!   [beep!]
.........       [beep!]
Encryption process successful!!!
Time taken for encryption is 29.000000 sec.
press any key to continue

Enter:
  '1' for encryption
  '2' for decryption
  '3' for Key generation
                                    <2>
Enter Your IDENTITY:                <BALU>
ID check O.K.
file to be deciphered =             <try.rsa>
output filename =                   <try.dec>
DECRYPTING!!!   [beep!]
.........       [beep!]
Time taken for decryption is 29.000000 sec.
Decryption process successful!!!
press any key to continue
```

A sample session for Key Generation:

```
Please enter the USER-ID you require in alphanumeric        <ASHA>
Enter the Key size in digits:                               <17>
The Public Key is:
n:  655447648672873633149324941930922 7
e:  459321802401257555071339977644 99
The Secret Key is:
d:  125235208348570482669174285178 7
p:  92793613865686541
q:  70634995380349847
#E_p(0,b):  92793613865686542
#E_q(0,b):  70634995380349848
N_n:    364137582596040916384547703630312
```

5.5 Conclusion

In this thesis we looked into various aspects of implementing a cryptosystem. We discussed few techniques for efficient computation with specific reference to elliptic curve cryptosystem. The effectiveness of these algorithms were tested through implementations and checked. The combination of Montgomery reduction with signed binary window method was found to be effective. Montgomery technique is effective in not only software but also in hardware. This has been studied in [5] and [22]. This would be of extreme importance in applications requiring high throughput like voice encryption etc. Through this thesis we have developed a platform for performing computation on elliptic curves. Also, encryption schemes on elliptic curves have been implemented and demonstrated. Even though the encryption rate is too low, the software implementation can be used to study the implementation over various curves.

Still lot more can be done for improving the computation on elliptic curves. In this thesis we have not looked into the implementation over Galois fields. The bottleneck in a high-speed implementation is the complex group algebra involved in the computation. Especially arithmetic in Galois field is time consuming and has to be efficiently implemented. The impact of various basis of representation of field elements has to be evaluated. Optimal normal basis has been found to be efficient for hardware implementations. But their impact on software implementation has to be studied. Oflate so many encryption schemes and signature schemes have been proposed. A study of these implementations and their suitability for implementation over elliptic curves can be examined. Another aspect which we have not covered here and which is most important is the design of elliptic curves suitable for cryptosystems.

Bibliography

[1] G. Agnew, T. Beth, R. Mullin and S. Vanstone, Arithmetic operations in GF(2m), Journal of Cryptology, 6(1993), 3-13.

[2] G. Agnew, R. Mullin, I. Onyszchuk and S. Vanstone, An implementation for a fast public-key cryptosystem, Journal of Cryptology, 3(1991), 63-79.

[3] G. Agnew, R. Mullin and S. Vanstone, An implementation of elliptic curve cryptosystems over F_{2155}, IEEE Journal on Selected Areas in Communications, to appear.

[4] Antoon Bosselaers, Rene Govaerts and Joos Vandewalle, Comparison of three modular reduction functions, CRYPTO '93.

[5] B. Arazi, Double-precision Modular Multiplication based on a Single-precision Modular Multiplier and a standard CPU, IEEE J. Select. Areas in commn., Vol.11, No. 5, June 1993, 761-769.

[6] Atsuko Miyaji, Elliptic Curves over F_p Suitable for Cryptosystems, AUSCRYPT '92, Lecture Notes in Computer Science, Springer-Verlag.

[7] Atsuko Miyaji, Elliptic Curve Cryptosystems Immune to Any Reduction into the Discrete Logarithm Problem, IEICE Trans. Fundamentals, Vol. E76-A, No. 1 January 1993.

[8] A. Bender and G. Castagnoli, On the implementation of Elliptic Curve Cryptosystems, Advanced in Cryptology - CRYPTO '89 Proceedings, Berlin: Springer-Verlag, 1990, pp 186-192.

[9] T. Beth and F. Schaefer, Non-Supersingular Elliptic Curves for Public Key Cryptosystems, Advances in Cryptology - EUROCRYPT '91 Proceedings, Berlin: Springer-Verlag, 1991, pp 316-327.

[10] Brassard, G. Modern Cryptology. Lecture Notes in Computer Science, Vol. 325. Springer-Verlag 1988.

[11] Bruce Schneier, Applied Cryptography, John Wiley & Sons Inc., New York

[12] Burton S. Kaliski, Jr., Multiple-precision Arithmetic in C, Dr. Dobb's Journal, August 1992.

[13] Burton S. Kaliski, Jr., The Z80180 and Big-number Arithmetic, Dr. Dobb's Journal, September 1993.

[14] Comba, P.G. Exponentiation Cryptosystems on the IBM PC. IBM Systems Journal, 29,4 (1990), pp 526-538.

[15] D.E. Denning, Cryptography and Data Security, Addison-Wesley, 1982.

[16] N. Demytko, A new elliptic curve based analogue of RSA, Proc. of Eurocrypt'93, pp. 39-48, May 24 (1993)

[17] W. Diffie, The First Ten Years of Public-Key Cryptography, Proceedings of the IEEE, Vol. 76, No. 5, May 1988, pp 560-577

[18] W. Diffie and M.E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, Vol. IT-22, No. 6, Nov. 1976, pp 644-654.

[19] Duncan A. Buell and Robert L. Ward, A Multiprecise integer arithmetic package, The Journal of Supercomputing 1989, 3:89-107.

[20] S. Dusse and D. Kaliski, A cryptographic library for the Motorola DSP56000, Advances in Cryptography - EUROCRYPT '90, Lecture Notes in Computer Science, 473 (1991), Springer-Verlag, 230-244.

[21] Duncan A. Buell, A Multiprecise Integer Arithmetic Package, The Journal of Supercomputing 3, 89-107 (1989).

[22] S.E. Eldridge and C.D. Walter, Hardware Implementation of Montgomery's Modular Multiplication Algorithm, IEEE Trans. on Comp., Vol. 42, No. 6, June 1993, 693-699.

[23] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete lograrithms, IEEE Transactions on Information Theory, 31 (1985), 469-472.

[24] G. Harper, A. Menezes and S. Vanstone, Public-key Cryptosystems with very small key lengths, Advances in Cryptology - EUROCRYPT'92.

[25] J. Hastad, On using RSA with Low Exponent in a Public Key Network, Proc. of Crypto'85, pp. 403-408 (1985)

[26] P. Ivey, S. Walker, J. Stern and S. Davidson, An ultra-high speed public key encryption processor, Proceedings of IEEE Custom Integrated Circuits Conference, Boston, 1992, 19.6.1 - 19.6.4.

[27] Jeffrey W. Hamilton, Basic Arithmetic with Infinite Integers, Dr. Dobb's Journal, January 1995.

[28] Kenji Koyama and Yukio Tsuruoka, A Signed Binary Window Method for Fast Computing over Elliptic Curves, IEICE Trans. Fundamentals, Vol. E76-A, No. 1, January 1993.

[29] Knuth, D.E. The Art of Computer Programming, Vol 1: Fundamental Algorithms. Addison-Wesley, Reading, Mass., 1973.

[30] Knuth, D.E. The Art of Computer Programming, Vol 2: Seminumerical Algorithms. Addison-Wesley, Reading, Mass., 1981.

[31] N. Koblitz, Elliptic Curve Cryptosystems, Mathematics of Computation, Vol. 48, No. 177, 1987, pp 203-209.

[32] N. Koblitz, Primality of the number of points on an elliptic curve over a finite field, Pacific Journal of Mathematics, 131 (1988), 157-165.

[33] N. Koblitz, Constructing Elliptic Curve Cryptosystems in Characteristic 2, Advances in Cryptology - CRYPTO '90 Proceedings, Berlin: Springer-Verlag, 1991, pp 156-167.

[34] Koblitz, N., A course in Number Theory and Cryptography, GTM114, Springer-Verlag, New York, 1987.

[35] K. Koyama, U.M. Maurer, T. Okamoto and S.A. Vanstone, New Public-key Schemes based on elliptic curves over the ring Z_n, Proc. of Crypto'91 (1991)

[36] H. Kuwakado and K. Koyama, Security of RSA-type cryptosystems over elliptic curves against Hastad attack, Electronics Letters, Vol. 30, No. 2, 27th October (1994)

[37] K. Kurosawa, K. Okada, S. Tsujii, Low Exponent Attack against Elliptic Curve RSA, ASIACRYPT '94, LNCS, Vol. 917, 376-383.

[38] A. Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, 1993.

[39] A. Menezes and S. Vanstone, A note on cyclic groups, finite fields, and the discrete logarithm problem, Applicable Algebra in Engineering Communication and Computing, 3(1992), 67-74.

[40] A. Menezes and S. Vanstone, Elliptic curve cryptosystems and their implementation, Journal of Cryptology, 6 (1993), 209-224.

[41] A. Menezes and S. Vanstone, The Implementation of Elliptic Curve Cryptosystems, AUSCRYPT '90, Lecture Notes in Computer Science, vol. 453, Springer-Verlag, 1-13.

[42] A. Menezes, S. Vanstone and R. Zuccherato, Counting points on elliptic curves over F_{2^m}, Mathematics of Computation, 60 (1993), 407-420.

[43] A. Menezes, T.Okamoto and S. Vanstone, Reducing elliptic curve logarithms to logarithms in afinite field, IEEE Transactions on Information Theory.

[44] Miller, V.S., Use of Elliptic Curves in Cryptography, Advances in Cryptology-Proceedings of Crypto '85, Lecture Notes in Computer Science, 218, Springer-Verlag, pp 417-426, 1986.

[45] Montgomery, P. Modular Multiplication Without Trial Division. Math. Comput., 44, (April 1985), 519-521

[46] Montgomery, P. Speeding the Pollard and Elliptic Curve Methods. Math. Comput., 48, (January 1987), 243-264

[47] Niven, Ivan and Zuckerman, H.S. An Introduction to the Theory of Numbers, Wiley-Eastern, Third Ed. 1976.

[48] Pollard, J.M. Monte Carlo Methods for Index Computation (mod p). Math. Comp. Vol. 32, No. 143, pp 918-924, 1978.

[49] Rivest, R., Shamir, A. and Adleman, L. A Method for obtaining Digital Signatures and Public-Key Cryptosystems. Comm. ACM, 21, 2 (Feb 1978), 120-126; reprinted in comm. ACM, 26, 1 (Jan 1983), 96-99.

[50] T. Rosati, A high speed data encryption processor for public key cryptography, Proceedings of IEEE Custom Integrated Circuits Conference, San Diego, 1989, 12.3.1 - 13.2.5.

[51] Silverman, J.H., The Arithmetic of Elliptic Curves, GTM106, Springer-Verlag, New York, 1986.

[52] G. Simmons (editor), Contemporary Cryptology: The science of Information Integrity, IEEE Press, New York, 1991.

[53] John T. Tate, The Arithmetic of Elliptic Curves, Inventiones Math., Vol. 23, pp 179-206 (1974).

[54] Y. Yacobi, Exponentiating Faster with Addition Chains, Proc. of EUROCRYPT '90, 1990.